AFRL-IF-RS-TR-2002-314
Final Technical Report
December 2002

MODELING ABSTRACTION AND SIMULATION TECHNIQUES

University of Massachusetts

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

APPROVED: [illegible]

TIMOTHY E. BUSCH
Project Engineer

FOR THE DIRECTOR: [illegible]

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** DECEMBER 2002	**3. REPORT TYPE AND DATES COVERED** Final Jun 99 – Apr 02

4. TITLE AND SUBTITLE MODELING ABSTRACTION AND SIMULATION TECHNIQUES	**5. FUNDING NUMBERS** C - F30602-C-99-0056/0057 PE - 62702F PR - 459S/459S TA - BA/BA WU - 94/91
6. AUTHOR(S) Christos G. Cassandras and Wei-Bo Gong	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Massachusetts (0056) Office of Grant & Contract Administration 408 Goodell Building, Box 33285 Boston MA 02215 Trustees of the Boston University(0057) 881 Commonwealth Avenue Boston MA 02215	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFSB 525 Brooks Road Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2002-314

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Timothy E. Busch/IFSB/(315) 330-1486/ Timothy.Busch@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

This final report contains the joint work of both University of Massachusetts and Boston University who were each funded under separate contracts. The objective of this effort has been to develop and study three novel complementary directions that may be summarized as follows:
1. Extract information from the inherently slow simulation process of complex systems by exploiting new concurrent simulation techniques.
2. Exploit the hierarchical structure in multi-resolution models by decomposing them in ways which preserve statistical fidelity.
3. Explore the use of neural networks as complex simulation metamodels. The scope of the project has been to develop specific methodologies and algorithms based on the proposed new techniques that were developed and tested. In many cases, The benchmark problems studied are the same or extensions of the ones developed during our previous projects "Enabling Technologies for Real-Time Simulation' and "Real-Time Simulation Technologies for Complex Systems.

14. SUBJECT TERMS Metamodeling, Model Abstraction Techniques, Concurrent Simulation	**15. NUMBER OF PAGES** 164
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

This report summarizes the work we have performed for the project entitled "Modeling Abstraction and Simulation Techniques." The objective of this effort has been to develop and study three novel complementary directions that may be summarized as follows:

1. Extract additional information from the inherently slow simulation process of complex systems by exploiting new *concurrent simulation* techniques.
2. Exploit the *hierarchical* structure in multi-resolution simulation models by decomposing them in ways which preserve *statistical fidelity*.
3. Explore the use of *neural networks* as complex simulation *metamodels*.

The scope of the project has been to develop specific methodologies and algorithms and test them on benchmark problems in C^4I application areas. Thus, appropriate simulation models were built, and algorithms based on the proposed new techniques were developed and tested. In many cases, the benchmark problems studied are the same or extensions of the ones developed during our previous projects "Enabling Technologies for Real-Time Simulation" [12] and "Real-Time Simulation Technologies for Complex Systems" [13].

We begin by briefly outlining some of the major challenges faced by modeling and simulation techniques for complex systems and the approaches we are following to address these challenges (Section 1.1). We then describe the organization of this report (Section 1.2).

1.1 Issues in Modeling and Simulation of Complex Systems

Simulation is widely recognized as one of the most versatile and general-purpose tools available today for modeling complex processes and systems and for solving problems in design, performance evaluation, decision making, and planning. In the C^4I environment, in particular, most situations that confront analysts and decision makers are of such complexity that handling them far surpasses the scope of available analytical and numerical methods; this leaves simulation as the only alternative of "universal" applicability. Unfortunately, there are several factors that limit the use of simulation.

1. For most situations of practical interest, simulation is extremely time-consuming.
2. In order to evaluate different alternatives, one has to perform a large number of simulations (one for each alternative). Furthermore, combining and processing the resulting data in a way which enhances decision making capability is a difficult task.

Since simulation is so time consuming, it is usually viewed as an *off-line* tool: one has to wait for the completion of one or more simulation runs before deciding how to interpret the results and how to proceed next. Our objective in this project is to transform simulation into a much more *interactive* tool not only for "evaluation" of alternatives, but also for efficient real-time "optimization" over alternatives. It is also desirable to utilize simulation as a means towards obtaining much simpler, yet accurate, surrogate models of the complex process or system of interest; this is also referred to as *metamodeling*.

To achieve the objectives outlined above, during this project we have pursued the following complementary directions

1. **Model abstraction using fluid simulation:** As already mentioned, Discrete-Event Simulation is time consuming and impractical due to the large number of events that are usually involved. An alternative *abstract* modeling paradigm is based on Fluid Models (FM). The fluid-flow worldview can provide either approximations to complex discrete-event models or primary models in their own right. Furthermore, fluid models can be combined with discrete-event models to develop a class of *hybrid-systems*, where, the state of the system is described by discrete as well as continuous type variables and the system dynamics are both, time-driven and event-driven. Such hybrid models can be used to model a fairly broad class of systems including battle engagements, communication networks, manufacturing systems and many more.

   The justification of FM rests on the realization that some events are more important than others. In effect, fluid models aggregate several events into a single event making simulation significantly more efficient. For example, in the context of high speed communication networks, the effect of an individual packet or cell on the entire traffic process is virtually infinitesimal, not unlike the effect of a water molecule on the water flow in a river. To appreciate the effectiveness of FM, consider for example a discrete event simulation run of an ATM link operating at 622 Megabits-per-second which requires the processing of over a million events per second. On the other hand, if traffic comes from the source at rates that are piecewise-constant functions of time, then a simulation run would process only one event per rate change. Thus, 30 rate changes per second (as in certain video encoders) may require the processing of only 30 events per second.

When using abstraction techniques (like the fluid models) it is important to determine the right *resolution* (level of abstraction), i.e., the number of events that are aggregated and treated as a single event. High resolution models (detailed simulation) are impractical. On the other hand, very low resolution implies significant approximation errors. In this report, we analyze the tradeoff between the fidelity of the simulation results and the resolution level of fluid simulation.

2. **Control and optimization using fluid simulation:** In a complementary direction, even in cases where the accuracy of a fluid model is not very high, the model might still be usable for the purpose of *control and optimization* rather than *performance analysis.* In this case, it is not unreasonable to expect that one can identify the solution of an optimization problem based on a model which captures only those features of the underlying "real" system that are needed to lead to the right solution, but not necessarily estimate the corresponding optimal performance with accuracy. Even if the exact solution cannot be obtained by such "lower-resolution" models, one can still obtain near-optimal points that exhibit robustness properties with respect to certain aspects of the model they are based on. Such observations have been made in several contexts (e.g., [63, 60, 17]).

3. **Concurrent simulation:** In simulation studies it is generally required to evaluate the performance $J(\cdot)$ of the system under a set of parameters/scenarios $\{\boldsymbol{\theta}_0, \cdots, \boldsymbol{\theta}_N\}$ (where each $\boldsymbol{\theta}_i$ is generally a vector quantity). The typical solution approach is to repeatedly simulate the system under each parameter/scenario which requires at least $N+1$ simulation runs. If a typical simulation requires T time units, then this process requires a total of $(N+1)T$ time units. In the context of *concurrent simulation* it is desired to extract additional information from a *single* simulation run under a parameter $\boldsymbol{\theta}_0$ besides the measure $J(\boldsymbol{\theta}_0)$. One possibility is to obtain $\{J(\boldsymbol{\theta}_0), \cdots, J(\boldsymbol{\theta}_N)\}$. In this case, it is also required that $(N+1)T >> T+c$ where c is the required computational overhead. Alternatively, one can obtain gradient information $\nabla J(\boldsymbol{\theta}) = [\frac{\partial J}{\theta_1}, \cdots, \frac{\partial J}{\theta_n}]$ which can be used together with stochastic optimization schemes [17] for optimization or to expedite the training of a neural network metamodel (see Chapter 4).

4. **Model abstraction using neural networks:** In the context of simulation, the main idea of metamodeling is to build a "surrogate" model of the system of interest which is much simpler (yet accurate) to work with. This is essentially analogous to constructing a function $F(\mathbf{x})$, $\mathbf{x} = [x_1, \cdots, x_N]$ from only a finite set of selected samples $\mathbf{x}^1, \cdots, \mathbf{x}^M$. The problem, of course, is that the actual function we are trying to approximate with $F(\mathbf{x})$ is unknown. There are several approaches to metamodeling, many of which are domain-specific (e.g., see [2, 29, 30]). The most common general approach is to try and build a polynomial expression. This is often inadequate because if the shape of the actual curve corresponding to $F(\mathbf{x})$ includes sudden jumps and asymptotic behavior (which is very often the case from experience), then polynomial fits to such curves are known to be poor.

   In our work we address what we view as two key challenges in the domain of metamodeling: (a) Maximizing the amount of information extracted from simulation so as to enhance the accuracy of the metamodel constructed and (b) Obtaining a metamodeling device of "universal" applicability, i.e., one capable of generating functions of virtually arbitrary complexity. The first challenge is addressed through the concurrent simulation approach discussed earlier. The second involves the use of neural networks as surrogate models. Neural networks have been used successfully in many areas of application (e.g., speech and pattern recognition) and have generated a great deal of enthusiasm for the promise they bring (e.g., see [27, 83, 53]). The main idea is to view a neural network as a device that acts as a "metamodel" and provides a desired response curve in great generality; that is precisely its strength. We have used several benchmark problems to date to compare neural networks to state-of-the-art alternatives and have obtained extremely positive results [12, 39, 14, 66]. Here we point out that the form of the additional information extracted through concurrent simulation is also important. For example, it is not immediately obvious how gradient information can be used in the metamodel development. This is investigated in [66] and Chapter 4 of this report.

5. **Hierarchical simulation and statistical fidelity:** One way to reduce complexity is through hierarchical decomposition of a simulation model. The challenge here is to do it without sacrificing accuracy. By "accuracy" we mean that the statistical information generated at the low-level, high-resolution simulation model should be preserved accurately at the higher-level models. In focusing on the preservation of the stochastic fidelity in hierarchical battle simulation models we have worked with a concrete model [12, 13] and analyzed various approaches for the preservation of stochastic fidelity. Our effort has been directed at developing an interface between the two simulation levels to preserve the statistics to the maximum extent that the available computing power allows. In our previous project [12] we initiated a study of an approach based on *clustering* or *path bundle grouping* which is further pursued in this project.

1.2 Organization

The content of this report is organized as follows.

Chapter 2: First we review the basic fluid simulation (FS) modeling framework and its variants: time stepped simulation (TSS), time driven fluid simulation (TDFS), and time stepped hybrid simulation (TSHS). Subsequently we investigate the "accuracy" of the fluid simulation models mainly by studying the errors generated by ignoring the detailed dynamics at small time scales. Through multiple simulation studies we demonstrate how the *resolution* affects the error of the abstract fluid model.

Chapter 3: We present the general framework of concurrent simulation. Subsequently, we adopt the stochastic fluid modeling (SFM) framework (a simple variant of the FS framework presented above) and based on it we derive sample derivative estimates of the performance measures of interest. Furthermore, these derivatives are very easy to evaluate and we show that they are unbiased and nonparametric, i.e., they do not depend on the stochastic processes that drive the system dynamics. Finally, we show that these derivatives can be evaluated directly from the sample path of the discrete event system. In other words, it is not necessary that we revert to a fluid simulation model, rather we can extract the sensitivity information directly from the discrete event simulator.

Chapter 4: We review the main concepts involved in using neural networks as universal function approximators. Training neural networks typically requires a large number of training points and since these points are obtained from simulation, the training process is very time-consuming. To speed up the data collection one can use concurrent simulation. It is straightforward to use additional information that is in the form of the system's output under different input parameters. On the other hand, it is not obvious how sensitivity information generated by concurrent simulation can be used. We investigate the use of sensitivity information to reduce the simulation effort required for training a NN metamodel.

Chapter 5: In this chapter, we discuss the applications of clustering methods in hierarchical simulation of complex systems and in system modelling. In the first part, we discuss the basic concepts for multi-resolution simulation modelling of complex stochastic systems. We argue that high-resolution output data should be classified into groups that match underlying patterns or features of the system behavior before sending group averages to the low-resolution modules to maintain the statistical fidelity. We propose high-dimensional data clustering as a key interfacing component between simulation modules with different resolutions and use unsupervised learning schemes to recover the patterns for the high-resolution simulation results. In the second part, we give the examples of using a Hidden Markov Model as an effective clustering tool for this task. Subsequently, we apply the clustering approach to a computer security problem (intrusion detection) and give examples of using Hidden Markov Models for the purpose of system modelling for anomaly detection.

Chapter 6: In this chapter we investigate the use of sensitivity information (obtained through concurrent estimation) together with stochastic approximation schemes for "real-time" optimization purposes. Our examples are derived from the areas of computer networks and mission planning in the context of Joint Air Operations (JAO).

Chapter 7: We present the main conclusions of our study, including lessons learned and recommendations. We also outline our ongoing work and some future research directions.

Chapter 2

MODEL ABSTRACTION USING FLUID MODELS

Modelling and performance evaluation are crucial the design, development and management of complex systems (e.g., computer networks). Conventional analytical methods usually rely on overly simplified assumptions while discrete event simulation (DES) is computationally prohibitive (requires long simulation runs). Thus the scalability of evaluation tools has been the focus of many studies and is also the topic of this chapter where we are motivated by problems in traffic engineering [5] as they apply in computer networks.

2.1 Introduction

Several directions can be followed to improve model scalability. Parallel DES, such as SSF [74] and *PDns* (parallel/distributed *ns*) [65], take advantage of the computational power of multiprocessors or distributed computer networks. Another way is to raise the abstraction level of modelling and simulation. For example, a fluid model only captures traffic burstiness at a large timescale. The resulting fluid simulation (FS) [47] tracks the fluid rate changes caused by sources and multiplexing at various nodes in the network. As mentioned in the previous chapter, since the frequency of rate changes is typically much lower than packet transmission rates, it is expected to achieve significant simulation speedup. However, due to the well-known ripple effect [55], FS may become more expensive than DES when evaluating large complex networks. Thus researchers use hybrid methods to achieve the quick simulation goal. For example, *Opnet* simulator [46] combines analytical techniques and DES.

In this chapter we study time-stepped-simulation (TSS), an abstract simulation scheme, where the time axis is discretized into small fixed intervals called time steps. The stochastic behavior of traffic within the time step is ignored and constant arrival rates are assumed. The simulation proceeds in a time-driven fashion, updating the system state periodically.

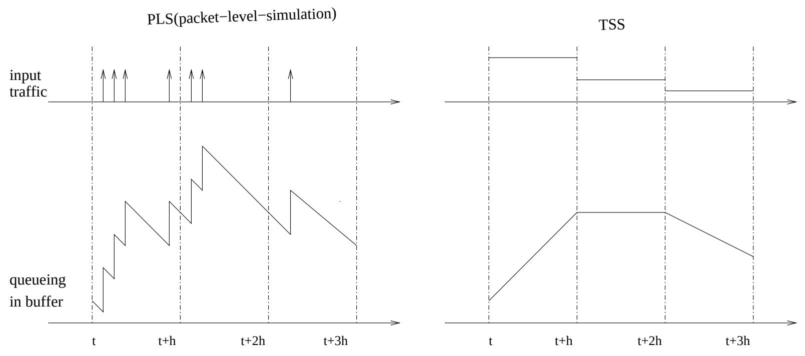


Figure 2.1: TSS and PLS in a queue (h denotes the time-step length).

Fig. 2.1 illustrates how TSS and Packet-level-simulation (PLS) view source traffic and the queueing evolution in the buffer.

TSS trades off accuracy for speedup, eliminates the FS ripple effects and is capable of adjusting simulation granularity to various levels. Moreover, TSS can facilitate parallel simulation due to its synchronization nature. Time driven fluid simulation (TDFS) [85] and time stepped hybrid simulation (TSHS) [38] also belong to this category. In TDFS the fluid rate during each interval is the average rate over that interval and queueing backlogs are tracked in discrete time. In order to simulate packet-based protocols such as TCP, Guo [38] proposes TSHS, where packets from the same session are assumed to be evenly spaced within the time-step.

Accuracy is one of major issues of abstract simulation where errors are mainly due to the following.

1. No packet inter-arrival variation within each time-step or flow state.
2. No discrete workload in fluid-type models.
3. Flow interference in multiplexors.
4. The procedure of extracting packet statistics (for example, inferring packet end-to-end delays from FS).

Yan [85] and Guo [38] give error bounds of TSS. This chapter focuses on simulation accuracy and mainly studies the error from ignoring traffic randomness at small timescales.

2.2 Impact of Autocorrelation

2.2.1 TSS for an $M/D/1$ Queue

Consider the simplest queueing system $M/D/1$. Assume that the service time of one packet is d. If TSS chooses abstraction level h, the simulation proceeds in time intervals of length

$h \times d$. Let x_n be the number of packets in the system at the beginning of time step n and a_n the number of arrivals during this interval. We have the following system equation

$$x_{n+1} = [x_n + a_n - h]^+. \tag{2.1}$$

where $[x]^+ = \max\{0, x\}$. We introduce an auxiliary random variable y_n

$$y_{n+1} = [y_n - h]^+ + a_n. \tag{2.2}$$

Assuming that the system is stable, let x, y and a denote the stationary value of x_n, y_n and a_n. We have

$$Ex = Ey - Ea. \tag{2.3}$$

Appendix A gives the derivation of Equation (2.3). $Y(z)$ and $A(z)$ are the probability generating functions (PGF) of y and a respectively. Equation (2.2) is the evolution equation of a discrete-time multi-server single queue denoted by $G/D/h$ [8]. For TSS of $M/D/1$, we get

$$Ey = Y'(z)|_{z=1} = \frac{1}{2(1-\rho)} - \frac{h}{2}(1-\rho) + \sum_{i=1}^{h-1}(1-z_i)^{-1}, \tag{2.4}$$

where ρ is the utilization of the system and z_is are the $(h-1)$ poles of (2.5) with $|z_i| < 1$.

$$z^h e^{\rho h(1-z)} - 1 = 0. \tag{2.5}$$

Combining (2.3) and (2.4), we obtain the average number of packets in the system for abstraction level h.

$$Ex = \frac{1}{2(1-\rho)} - \frac{h}{2}(1+\rho) + \sum_{i=1}^{h-1}(1-z_i)^{-1} . \tag{2.6}$$

Let Eq denote the theoretical mean number of packets in an $M/D/1$ system.

$$Eq = \frac{1}{2(1-\rho)} - \frac{1-\rho}{2} . \tag{2.7}$$

Then, the TSS absolute abstraction error $(\Delta q(h,\rho))$ for abstraction level h and utilization ρ is given by:

$$\Delta q(h,\rho) = Eq - Ex = \frac{h-1}{2} + \frac{h+1}{2}\rho - \sum_{i=1}^{h-1}(1-z_i)^{-1}. \tag{2.8}$$

Finally, we also define the relative error $l\,(h,\rho)$ as

$$l(h,\rho) = \frac{\Delta q(h,\rho)}{Eq} = \frac{\frac{h-1}{2} + \frac{h+1}{2}\rho - \sum_{i=1}^{h-1}(1-z_i)^{-1}}{\frac{1}{2(1-\rho)} - \frac{1-\rho}{2}}, \tag{2.9}$$

where all z_i's can be obtained using numerical techniques. The simulation errors are shown in Fig. 2.2 and we make the following observations.

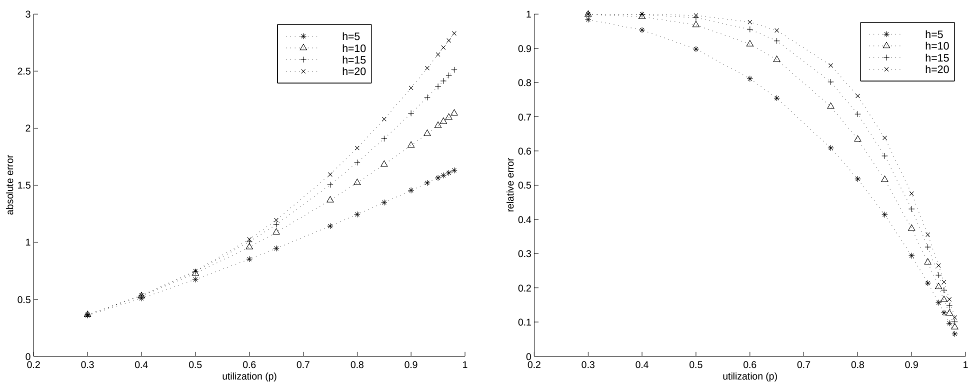


Figure 2.2: Absolute and relative simulation errors in mean system time for $M/D/1$.

Table 2.1: Relative simulation errors for $M/D/1$ ($h = 5$).

ρ	0.60	0.65	0.75	0.80	0.85	0.90
Err	0.8112	0.7544	0.6089	0.5182	0.4138	0.2939

1. The simulation accuracy is strongly related to the system utilization. The absolute error increases while the relative error decreases as utilization increases (see Fig. 2.2). As utilization approaches 1, the relative error reduces to zero which implies that low-resolution simulation works better for a heavily loaded system.

2. Low resolution is inaccurate but as utilization approaches the extremes, very low or very high, the relative error difference among different resolutions is small.

3. The utilization of practical networks often stays in the range of $[0.6 \sim 0.9]$. Table 2.1 gives the relative errors for this range. The simulation errors are big when the abstraction level $h = 5$. In the following, we will show that correlations in the source traffic make TSS work better.

2.2.2 Fluid Queue

Next we investigate a single fluid queue with piecewise constant inflow rates. The finest time scale of this flow is called frame. We first show experimentally that correlation in the source traffic results in better simulation accuracy. Then we provide an analytical justification.

Experiments

Correlated fluid processes are generated in a two-step synthetic method: (a) a basic process is cut into blocks containing B frames each, and (b) the blocks are randomly shuffled while the frame order in every block remains unchanged. The resulting processes have the same first order statistics as the basic one but different correlation. Fig. 2.3**a** shows the

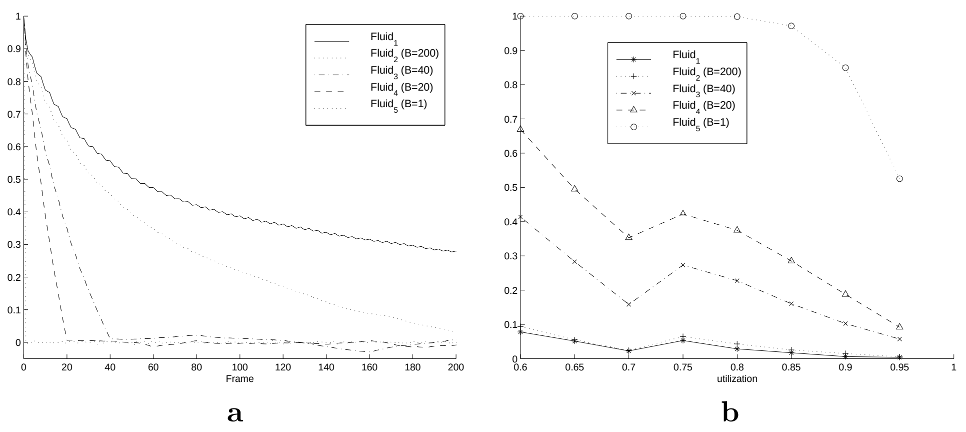


Figure 2.3: (**a**) Autocovariance coefficient curves of fluid processes. (**b**) Relative simulation errors in mean system time of fluid processes ($h = 50$).

autocovariance coefficient curves of 5 fluid sources. $Fluid_1$ is the basic one, which has the strongest correlation within the longest range. $Fluid_5$, generated by completely shuffling $Fluid_1$, is almost white noise. Each of these flows is used as an input to an infinite buffer queue which is simulated under two abstraction levels $h = 1$ and $h = 50$ ($h = 1$ is the simulation with the finest resolution). The two sets of simulation results are compared in Fig. 2.3**b** which shows the relative error for each input process. Note that the higher the inflow correlation, the smaller the approximation error while for $Fluid_5$ (uncorrelated traffic), the simulation accuracy is low even for high utilization system. This is consistent with the observations in the $M/D/1$ system.

How Correlation Works

TSS uses traffic abstraction and feeds approximate inputs into queues. Intuitively, the more an approximate input looks like the true input, the better the simulation accuracy. So the impact of correlation on simulation performance is checked based on how correlation contributes to the similarity of two input flows. Let x_i, $i = 1, 2, \cdots$, be a stationary process that denotes the fluid rate of the ith frame and define the following parameters:

1. $Ee_k^2 = E[(x_k - \frac{\Sigma_{i=1}^h x_i}{h})^2]$, k = 1,2,...,h, is the mean square of the error of the kth frame within each time step and h is the abstraction level. If Ee_k^2 is small, both the mean and variance of the error are small because $Ee_k^2 = Var|e_k| + (E|e_k|)^2$. This variable indicates how close the kth frames are in the original and approximate inflows.

2. $F = \frac{1}{h}\sum_{k=1}^h Ee_k^2$, is the total abstraction error that counts all frame errors in each time step. F is a general denotation. F_{un} and F_{re} are particular denotations for uncorrelated and correlated processes respectively.

It is easy to get

$$Ee_k^2 \quad = \quad \frac{Varx}{h^2} \sum_{j=1, j\neq k}^{h} \sum_{i=1, i\neq k}^{h} \Big(1 + \rho_{|i-j|} - \rho_{|k-i|} - \rho_{|k-j|}\Big) \ . \tag{2.10}$$

where ρ and $Varx$ are the autocovariance coefficients and the variance respectively. Uncorrelated processes have the following autocovariance coefficients

$$\rho_n = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \ . \end{cases} \tag{2.11}$$

and therefore, (2.10) reduces to

$$Ee_k^2 \quad = \quad Varx\left(1 - \frac{1}{h}\right) .$$

Then

$$F_{un} \quad = \quad Varx\left(1 - \frac{1}{h}\right) .$$

Appendix A derives the abstraction error for a correlated process.

$$F_{re} = F_{un} - C \ . \tag{2.12}$$

where

$$C \quad = \quad 2\frac{Varx}{h^2}\sum_{i=1}^{h-1}(h-i)\rho_i \ . \tag{2.13}$$

Discussion:

1. Combining Eqs. (2.13) and (2.12) we see that the abstraction error of the correlated process is smaller than that of the corresponding uncorrelated process if the two inputs have the same first order statistics. Also, the larger C is, the smaller the abstraction error and so the better the simulation accuracy.

2. According to Eq. (2.13), autocovariance coefficients in the shorter range are given larger weights. Better simulation performance is expected for fluid traffic with stronger correlations in short range. We conjecture that TSS with abstraction level h well simulates a single fluid queue if its input has strong correlation up to h frames. Table 2.2 gives the errors of the 5 fluid processes mentioned in Sect. 2.2.2. It shows that abstraction error increases as correlation decreases.

3. The variance of the rate of the fluid process also affects the simulation performance as seen in Eq. (2.13), where C is proportional to this variance.

It is more suitable to describe the traffic of computer networks by point processes. The impact of autocorrelation is also confirmed by experiments where more general queueing systems with point processes are investigated. Please refer to [84] for details.

Table 2.2: Statistical errors in input fluid processes brought by coarse simulation ($h = 10$)

Processes Approximated	$Fluid_1$	$Fluid_2$	$Fluid_3$	$Fluid_4$	$Fluid_5$
$E|e|$ (Experimental)	0.2988	0.2988	0.5858	0.7669	1.0733
F (Experimental)	0.4102	0.4102	0.8103	1.0661	1.5139
F(Analytical)	0.4105	0.4900	0.8207	1.0874	1.5137

2.3 Short-term and Long-term Traffic Characteristics

The previous section observes the impact of traffic correlation on the accuracy of time step simulation. This section uses sample path analysis to justify the impact of correlation and discusses the impact of burstiness at different levels.

2.3.1 Theoretical Explanation

Consider a single-server queue with a constant service rate, where the buffer is assumed infinite unless other specified. Cut the queueing sample path into blocks of length h. Each queueing block contains packet arrival information, the service rate and the initial backlog. Fig. 2.1 shows some queueing blocks. Nonempty queueing blocks are classified into two categories. (a) Fully busy blocks where the buffer is never empty during the block and, (b) partially busy blocks where the buffer is empty for some time. A busy train is a succession of fully busy blocks. Also, define the following parameters for a busy train of n time steps.

- μ: constant service rate
- $t_i = ih$, $i \in \{0, 1, \cdots, n\}$
- Q_{iF}: average queue length during $[t_{i-1}, t_i)$ in fine-time-scale simulation.
- Q_{iT}: average queue length during $[t_{i-1}, t_i)$ in time-step simulation.
- $a_i(t)$: rate function during $[t_{i-1}, t_i)$
- $\bar{a}_i$: average rate during $[t_{i-1}, t_i)$
- Q_{BF}: average queue length during the busy period $[0, nh]$ in fine-time-scale simulation.
- Q_{BT}: average queue length during the busy period $[0, nh]$ in time-step simulation.

First we assume that TSS has the same initial backlog as the fine-time-scale simulation as shown in Fig. 2.4**a**. A fully busy block is shown in Fig. 2.4**b** where the solid line demonstrates the fluctuation of the buffered workload in fine-scale simulation. Also, define

- QA_F: Queueing area in fine-time-scale simulation
- QA_T: Queueing area in time-step simulation

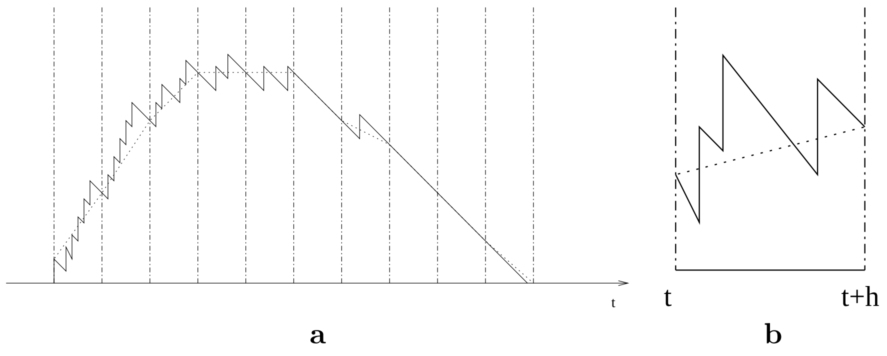


Figure 2.4: (**a**) A busy train. (**b**) A fully busy queueing block.

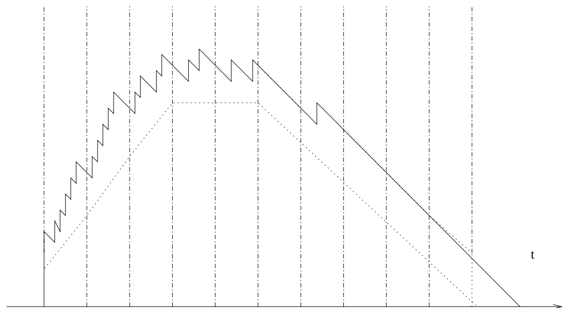


Figure 2.5: A busy train with initial discrepancy between TSS and PLS.

It is easy to get

$$\begin{aligned} QA_F &= QA_T + \int_0^h t\bar{a}dt - \int_0^h ta(t)dt \\ &= QA_T - \int_0^h tb(t)dt \end{aligned}$$

where $b(t) = a(t) - \bar{a}$ is the rate difference function. Therefore,

$$Q_F = Q_T - \frac{1}{h}\int_0^h tb(t)dt \tag{2.14}$$

where,

$$\int_0^h b(t)dt = 0, \qquad b(t) \geq -\bar{a}$$

Then for a busy train,

$$Q_{BF} = Q_{BT} - \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}\int_0^h tb_i(t)dt \tag{2.15}$$

where $b_i(t)$ is the rate difference function during $[t_{i-1}, t_i]$.

If TSS does not have the same initial backlog as the fine-time-scale simulation (as in Fig. 2.5), we have

$$Q_F = Q_T - \frac{1}{h}\int_0^h t b_i(t)dt + \Delta \tag{2.16}$$

where $\Delta \leq \mu h$ is determined by the backlog difference at the time step before the busy train.

Subsequently, we use queueing blocks as a comparison unit and define the simulation error of queueing block i as

$$e_i = \frac{|Q_{iF} - Q_{iT}|}{Q_{iF}}. \tag{2.17}$$

Therefore, the simulation error for the whole trace with N nonempty blocks is

$$e = \frac{1}{N}\sum_{i=1}^{N} e_i \tag{2.18}$$

Note that Q_{iT} of partially busy blocks is much smaller than Q_{iF}, thus $e_i \approx 1$. Let α denote the ratio of the fully busy blocks among the total nonempty blocks. Then the simulation error of the whole trace is expressed as

$$e \simeq (1-\alpha)\times 1 + \alpha\overline{\frac{\frac{1}{h}\int_0^h tb(t)dt + \Delta}{Q_F}} \tag{2.19}$$

Discussion:

1. The error in partially busy blocks is usually larger than that of fully busy blocks. Long busy periods increase the percentage of fully busy blocks α and help to reduce the total simulation error. A heavily loaded system has a large percentage of fully busy blocks. Thus TSS works generally well for this kind of system. For the mildly or lightly loaded systems, the performance of TSS depends on the traffic's characteristics. Under the same system utilization, long busy periods shows up only for some cases.

2. As Fig. 2.4 and Fig. 2.5 show, the large backlog during busy periods lessens the impacts of local rate variation and initial backlog discrepancy. In a busy train of n blocks where the initial backlog is assumed zero, Q_{BT} is decided by $\{\bar{a}_i,\ \ i = 1, \cdots, n\}$

$$Q_{BT} = \frac{h}{n}\sum_{i=1}^{n-1}[(n-i)\bar{a}_i] - \frac{h(n-1)}{2}\mu \tag{2.20}$$

$$\sum_{i=1}^{n}\bar{a}_i = n\mu \tag{2.21}$$

$$\bar{a}_i \geq 0 \tag{2.22}$$

In (2.20), the weights given to $\bar{a}_i$ decrease as i increases. Combining constraints (2.21) and (2.22), it is induced that the higher arrival rates at the beginning of the busy period and the lower rates at the end lead to larger Q_{BT}. Also, the larger the

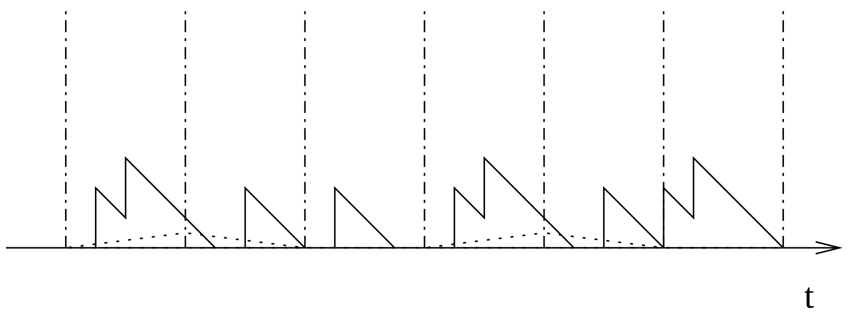


Figure 2.6: TSS poorly captures queueing dynamics.

rate difference between the two ends, the larger Q_{BT} is. At time scale h, if traffic is characterized by the concentration of periods with much higher arrival rates and the concentration of periods with very low arrival rate, it is easy to have large queueing building-up which results in reasonable simulation accuracy.

So far we have showed that TSS abstraction errors are due to queueing nonlinearity and rate variation during each time step. Thus, to reasonably evaluate a queueing process via TSS, the queueing process is required to show (a) long busy periods thus, reduce queueing nonlinearity, and (b) large queueing length during busy periods thus, reduce the impact of local rate variation

A heavily-loaded system generally satisfies the above two requirements. For a mildly-loaded system, acceptable accuracy requires that traffic shows long bursts with strong bursty intensity. Otherwise, TSS can not reflect real queueing process as shown in Fig. 2.6.

2.3.2 Experiments

By looking into the sample path of queueing process, we identify the factors that influence simulation accuracy and give analytical explanation. Here we show some experimental results to confirm our analysis. In the following discussion, *short-term* refers to the dynamics during a time step (how arrivals are spaced). *Long-term* refers to a time scale equal to or larger than the time step. In this study, we are mainly interested in the first and second order statistics of long-term traffic variability.

Traffic Model

We use a Hierarchical On-Off Modulated Poisson Process (HOMPP), which is based on the Hierarchical On-Off Process (HOP) [61]. An n-level HOP $Y(t)$ is defined as:

$$Y(t) = \prod_{1}^{n} X_i(t) \tag{2.23}$$

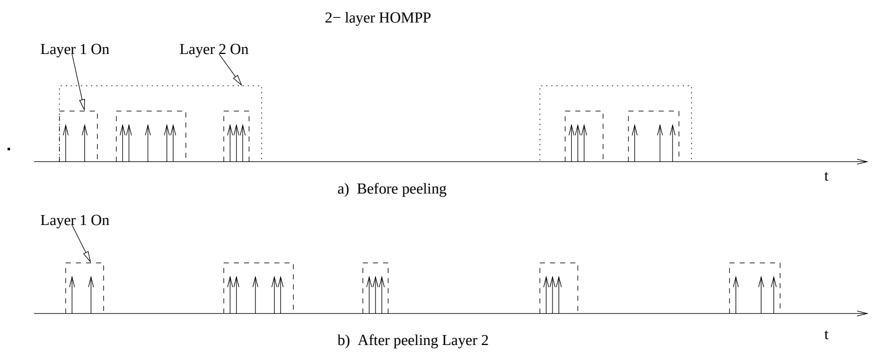


Figure 2.7: Peeling operation on HOMPP models.

where each $X_i(t)$ is an independent On-Off process. $Y(t)$ is in the "on" state only if all component processes are "on". HOMPP is generated from HOP $Y(t)$ by modulating a Poisson process. In this study, the timescales of different layers are disparate and Layer n works on the largest timescale. Any layer's "on" periods are concentrated within the "on" periods of its direct parent layer. This hierarchical model introduces burstiness at different timescales. We also define the "*peel*" operation at Layer i where some of the "on" periods of Layer $i-1$, rather than occurring during the "on" period of Layer i, they are uniformly distributed over the time axis (with no overlap) as shown in Fig. 2.7.

A HOMPP can model several levels of burstiness while peeling can adjust the degree of burstiness. HOMPP is used to investigate the impact of traffic dynamics at different timescales on TSS accuracy.

Long Term Dynamics

We compare simulation performance of hierarchical traffic. Table 2.3 lists the characteristics of three traffic sources. $H1$ has three on-off layers. On and off periods are independent and exponentially distributed. On average, "on" periods in Layer i contain five "on" periods in Layer $i-1$. $H1P3$ is generated by peeling $H1$'s Layer 3 and $H1P2$ by peeling Layers 3 and 2. The three traffic sources exhibit different burstiness properties. Let the time step h, and the average "on" period length of Layer 2 be equal to 25. Next, we investigate TSS performance and source characteristics at this time scale. Fluid trunks with high arrival workload in $H1P3$, is less concentrated than that of $H1$ because $H1P3$ lacks Layer 3. As expected, in this time scale, $H1P3$ has the same marginal distribution and less correlation as shown in Fig. 2.8**a** and Fig. 2.8**b**. In Fig. 2.9, $H1P3$ has worse simulation performance than $H1$ because the queueing due to workload accumulation in larger time scale is less than that of $H1$.

Next we compare the traffic characteristics of $H1$ and $H1P2$. In Fig. 2.8**b**, $H1P2$ lacks variability in its marginal distribution while arrivals in $H1$ are more widely spread. $H1P2$ only has one on-off layer and the "on" periods are on average 1 second long, uniformly spaced on the time axis. So when observing this trace at a time scale of 25 seconds, the

Table 2.3: Parameters of three HOMPP sources.

Traffic	Layer 0 Poisson rate(pkts/sec)	Layer 1: mean on/off length(sec)	Layer 2: mean on/off length(sec)	Layer 3:mean on/off length(sec)
$H1$	100	1 / 4	25 / 100	625 / 2500
$H1P3$	100	1 / 4	25 / 475	–
$H1P2$	100	1 / 88	–	–

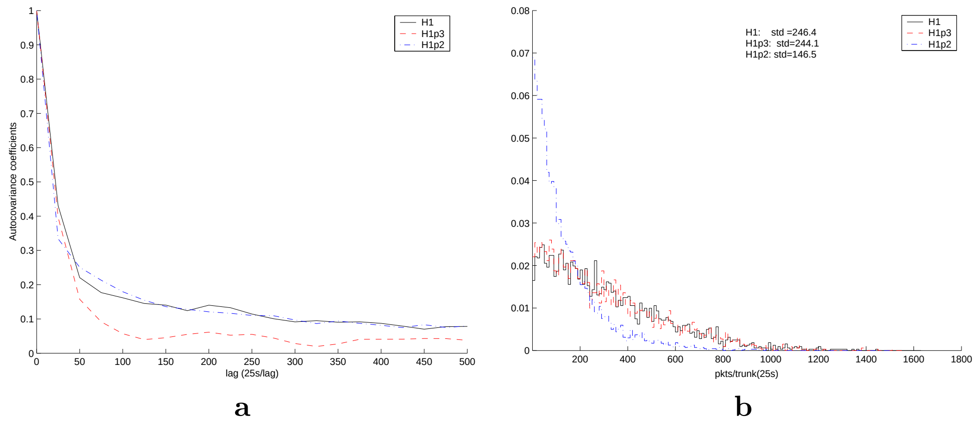


Figure 2.8: (**a**) HOMPP traffic autocovariance coefficient curves. (**b**) Marginal distribution of H1 and derived traffic (time step 25s).

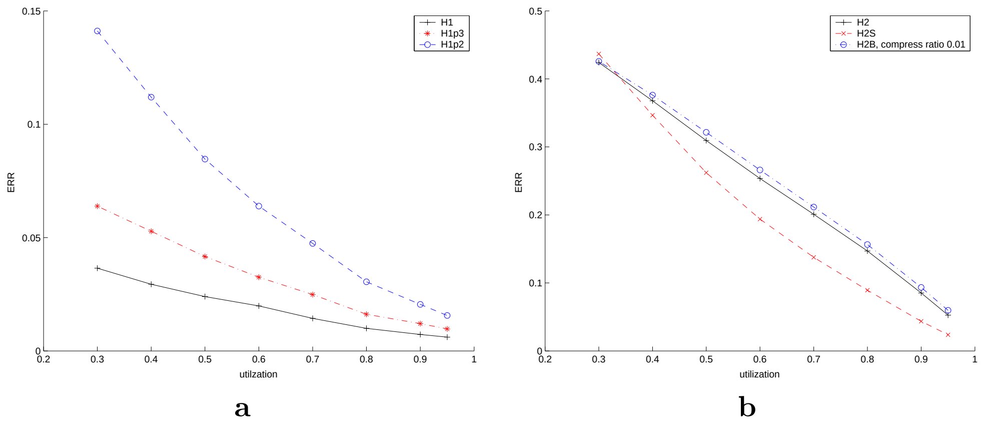


Figure 2.9: (**a**) Simulation errors of $H1$ and derived traffic (time step 25s). (**b**) Simulation errors of $H2$ and derived traffic (time step 5s).

trunk workload fluctuates around the average value in a small range. The workload amount in consecutive trunks is close and there is still correlation among trunks as shown in Fig.2.8**a**. According to the first order statistics, $H1P2$ is short of long term dynamics compared to $H1$. Fig. 2.9**a** shows the simulation error where, as expected, $H1$ outperforms $H1P3$ due to the rich dynamics in long-term scale. We emphasize that correlation can not be used as the only indicator for long-term dynamics. The first order statistics should also be considered. Take an extreme case for example. At certain time scale, traffic has constant average arrival rate and rate variations within time steps. TSS can not simulate this queue at this time scale. Even though this trace has strong correlation, it lacks long-term dynamics.

Short-Term Burstiness

If variability in large time scales exists in input traffic, arrivals in long term are the major factor for queue building. This case favors TSS. However, local burstiness degrades TSS simulation performance. $H2$ is a 2-layer HOMPP with exponentially distributed on-off periods. On Layer 1, the average "on" and "off" periods are 1s and 4s respectively. On Layer 2, they are respectively 25s and 125s. Set the time step equal to 5s. It is expected that arrivals most likely cluster within 1/5 of the interval. Smooth $H2$ to generate a new source $H2S$. That is, if there are n arrivals in a time step, place these arrivals according to uniform distribution over the interval. Therefore, $H2S$ is less bursty than $H2$, and shows an improved accuracy as shown in Fig. 2.9**b**.

The next experiment demonstrates what will happen if we make $H2$ more locally bursty. Change $H2$ local arrival pattern. If there are n arrivals in a time step, the arrivals are placed uniformly in part of the time step. The compression ratio determines how local bursty the traffic is. $H2B$ is the derived source of $H2$ with compression ratio 0.01. Its simulation accuracy does not degrade, compared to that of $H2$, as shown in Fig. 2.9**b**.

To study the impact of local burstiness when queueing in large time scale does not dominate, we use external shuffling to generate a new source which lacks strong dynamics in long term. $H2$ is chopped into blocks of fixed length and then the blocks are randomly permuted while keeping the relative position of arrivals within a block unchanged. Choose block length $0.1s$. The new source, $H2EXT$, is less bursty than $H2$ in long term. Then we compare $H2EXT$ with its derived traffic with compression ratio 0.5 and 0.1. As show in Fig. 2.10, local burstiness has an impact.

The above experiments show that TSS feasibility strongly depends on the traffic characteristics and the system utilization. For a given time scale, if source traffic does not show significant long-term dynamics (long bursts with enough intensity), TSS at the corresponding time step works poorly unless the system is heavily loaded.

2.3.3 Compensation

According to the previous discussion, local queueing dynamics should be counted to improve simulation accuracy. Assume that local statistics are known, but the exact arrival steps are

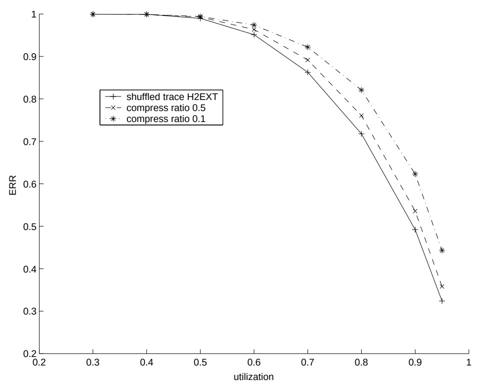


Figure 2.10: Simulation errors of $H2EXT$ and its derived traffic (time step $5s$).

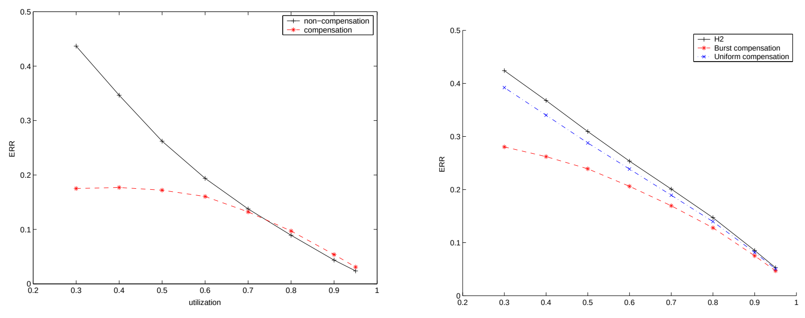
unknown. We use TSS to track the queue evolution, and for every time step, add $\overline{q}_{loc}$ to $\overline{q}_t$ to compensate local queueing dynamics.

In a time step, given the traffic local statistics, the utilization ρ, zero initial backlog, and average $\overline{q(\xi)}$ we can determine $\overline{q}_{loc}$, where $\overline{q(\xi)}$ is the queue length averaged over the interval provided the arrival sample ξ follows the local statistics. Use off-line simulation to get the local queueing curve $\overline{q}_{loc} \sim \rho$. Then TSS uses this curve and adds $\overline{q}_{loc}(\rho)$ to $\overline{q}_t$ to account for the local queueing effects.

In the following, we experimentally show the performance of the compensation scheme. As mentioned before, $H2S$ arrivals are uniformly distributed within five-second time steps. Knowing that, we get $\overline{q}_{loc} \sim \rho$ curves. Fig. 2.11**a** shows the improved simulation accuracy, especially in low utilization situations. As mentioned before, $H2$ is a 2-layer Hierarchical source and the average "on" in Layer 2 is 25 seconds. Within this time scale, the local dynamics are governed by Layer 1 on-off modulation. Fig. 2.11**b** shows the results of two compensation schemes. One scheme assumes arrivals are uniformly distributed within the interval while the other one takes the burstiness resulted by Layer 1 modulation, into consideration. It shows that more accurate information for local statistics helps the compensation performance.

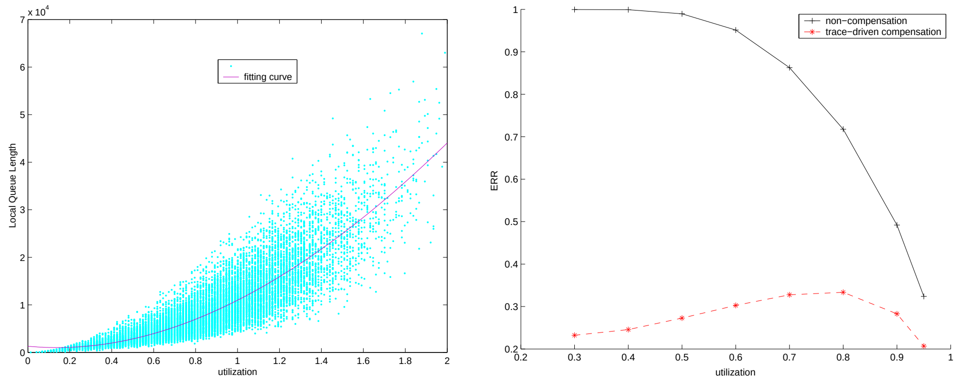
When local statistical information is unknown, we use trace-driven methods to extract local statistics assuming that local statistics are stable for the whole simulation. Feed the trace into a queue. Assuming zero backlog at the beginning of every time step, record the average queue length and corresponding utilization for every time step. In the plane of $\overline{q}_{loc} \sim \rho$, every pair of records is a point and we use curve fitting scheme to get the local queueing curve. Fig. 2.12**a** shows such curve for the source $H2EXT$. Fig. 2.12**b**shows the compensation results.

Combining local traffic statistics into TSS makes it work under a broad set of conditions. Our methods that get local queueing curves are quite rough and preliminary, but they emphasize that properly transforming local statistical information into macroscopic level simulation improves accuracy. This scheme provides a basis for improvement. In reality, local traffic statistics are not expected to change frequently and rapidly thus they can



a **b**

Figure 2.11: (**a**) $H2S$ compensation results (time step $5s$). (**b**) $H2$ compensation results (time step $5s$).



a **b**

Figure 2.12: (**a**) Local queueing curve of $H2EXT$ (time step $5s$). (**b**) Compensation results of $H2EXT$ (time step $5s$).

be measured on-line. Moreover, under certain scenarios, there are clues for traffic local statistical properties. For example, Cao et al. [9] observe that packet inter-arrival times tend to become independent as the number of active connections increases due to the statistical multiplexing. For this case, the compensation with uniform distribution is expected to work.

Discussion

David Nicol et al. [64] observe small simulation errors when comparing fluid and packet level simulation. Their traffic model is a Markov Modulated Process (MMP) and packets are transmitted at a specified constant rate when the underlying Markov chain is in some state. So their study does not consider the simulation errors from local traffic dynamics. Instead, they investigate the errors resulted due to the lack of workload discretization and flow interference.

Yan [85] derived lower and upper error bounds of TSS. For a single-flow single-server queue, the distance between two bounds is the time step multiplied by service rate. The bounds are tight only when queue building is mainly due to inter-trunk workload interaction. Otherwise, the bound is too loose to be useful because the bound distance is of the order of the actual queue length. This study helps in determining when bounds are tight.

This study raises several questions: when adjusting the system's granularity, from fine to coarse, how to properly abstract the component's micro behavior into a macroscopic one. In the single queue case, is it always proper to assume smooth/deterministic microscopic behavior of traffic? As our compensation experiments show, combining microscopic statistics into the abstract simulation expands simulation-working range.

This research also raises the issue of resolution in traffic modelling. We expect that for the performance evaluation of queueing systems, unless rare events are evaluated, it does not help much to model the rich local statistics at the cost of expensive modelling if traffic shows strong dynamics in larger timescales. This is because the impact of local traffic dynamics on queue is weakened by strong long-term-dynamics in source traffic. However, fine level traffic models such as multi-scaling [28] could be helpful in other evaluation scenarios.

2.4 Summary

Time stepped simulation can vary the abstraction levels and point out critical parts in a network design at a low modelling and simulation cost. Current results are encouraging. However, in practice, under what kind of scenarios can TSS work? We focus on the accuracy analysis of a single-flow single-server queue. We identify that the system utilization and traffic characteristics at short-term and long-term timescales affect the simulation accuracy. Queueing nonlinearity and local rate variation are two basic error sources for the considered scenarios. Therefore, we propose compensated TSS to combine local statistical information into TSS. Our results are encouraging.

Naturally this study will be expanded into networks of queues and study the effects of multi-flow interference and network topology. We plan to study the impact of flow and spatial granularity in addition to time granularity. In addition, Poisson-driven differential equations are a powerful tool for solving some queueing problems. We are using this tool to solve queueing systems that use Markov-hierarchy-onoff fluid input flows. This will provide insights on multi-resolution modelling errors.

Chapter 3

CONCURRENT SIMULATION

It is by now well-documented in the literature that the nature of sample paths of DES can be exploited so as to extract a significant amount of information, beyond merely an estimate of $J(\theta)$. It has been shown that observing a sample path under some parameter value θ allows us to efficiently obtain estimates of derivatives of the form $dJ/d\theta$ which are in many cases unbiased and strongly consistent (e.g., see [20, 31, 41] where Infinitesimal Perturbation Analysis (IPA) and its extensions are described). Similarly, Finite Perturbation Analysis (FPA) has been used to estimate finite differences of the form $\Delta J(\Delta\theta)$ or to approximate the derivative $dJ/d\theta$ through $\Delta J/\Delta\theta$ when other PA techniques fail [21].

All of the methods developed to date, regardless of specific details, have been motivated by the same objective: From a single sample path under θ extract information to estimate the derivative $dJ/d\theta$ or the response of the system, $J(\theta')$, under other parameter values $\theta' \neq \theta$ (see Fig. 3.1). This information can be extremely useful in sensitivity analysis and optimization of DES as well as data collection for metamodel building. Both of these applications will be demonstrated later in this report. Next we demonstrate the IPA approach using an example from the area of communication networks (for more details see also [17, 16]).

3.1 Introduction

A natural modeling framework for packet-based communication networks is provided through queueing systems. However, the huge traffic volume that networks are supporting today makes such models highly impractical. It may be impossible, for example, to simulate at the packet level a network slated to transport packets at gigabit-per-second rates. If, on the other hand, we are to resort to analytical techniques from classical queueing theory, we find that traditional traffic models, largely based on Poisson processes, need to be replaced by more sophisticated stochastic processes that capture the bursty nature of realistic traffic; in addition, we need to explicitly model buffer overflow phenomena which typically defy tractable analytical derivations.

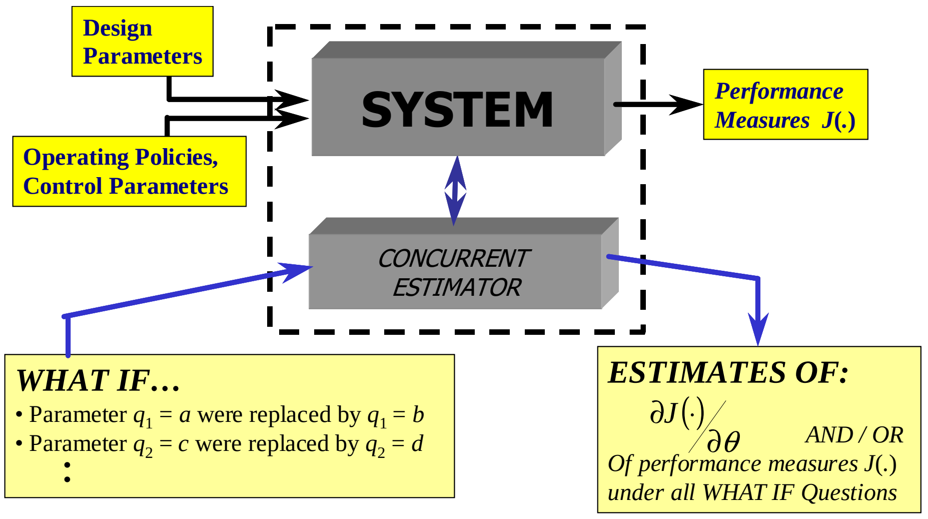

Design Parameters
Operating Policies, Control Parameters
SYSTEM
Performance Measures J(.)
CONCURRENT ESTIMATOR
WHAT IF…
• Parameter $q_1 = a$ were replaced by $q_1 = b$
• Parameter $q_2 = c$ were replaced by $q_2 = d$
ESTIMATES OF:
$\partial J(\cdot) / \partial \theta$
AND / OR
Of performance measures J(.) under all WHAT IF Questions

Figure 3.1: Concurrent Simulation Principle.

An alternative modeling paradigm, based on Stochastic Fluid Models (SFM), has been recently considered for the purpose of analysis and simulation [4, 48, 77, 47, 49, 62, 56, 86, 79]. The fluid-flow worldview can provide either approximations to complex discrete-event models or primary models in their own right. In any event, its justification rests on a molecular view of packets in moderate-to-heavy loads over high-speed transmission links, where the effect of an individual packet or cell on the entire traffic process is virtually infinitesimal, not unlike the effect of a water molecule on the water flow in a river.

Our objective in this chapter is no different from other perturbation analysis techniques: *From a single sample path under θ extract additional information to estimate the derivative $dJ/d\theta$.* In the discrete-event framework such derivative estimates are often biased. To avoid this problem, we adopt a stochastic fluid model and derive remarkably simple sensitivity estimators. These estimators turn out to be *nonparametric* in the sense that they are computable from data directly observable along a sample path, requiring no knowledge of the underlying probability law, including distributions of the random processes involved, or even parameters such as traffic or processing rates. In addition, the estimators obtained are unbiased under very weak structural assumptions on the defining traffic processes. Finally, because these estimators are non-parametric we can evaluate them based on data observed from the sample path of the discrete-event system, thus we do not necessarily need to construct the stochastic fluid equivalent model. In effect, we use the SFM only for the analysis part where we derive the structure of the IPA derivative estimators. However, when we actually evaluate them we simply observe the sample path of the discrete event system; either the true system or a discrete-event simulator.

The IPA gradient estimators that we derive can be readily used for on-line control purposes. For example, in the context of communication networks they can be used to perform periodic network management functions in order to guarantee negotiated QoS parameters and to improve performance. One such example is presented in Chapter 6 (see also [17, 16]

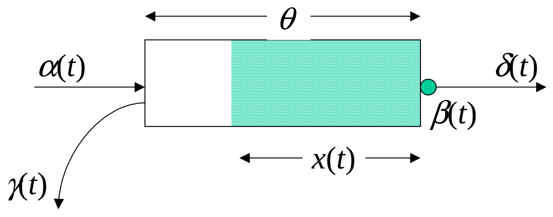


Figure 3.2: The basic Stochastic Fluid Model (SFM)

for more details). Aside from solving explicit optimization problems, IPA gradient estimators can be used for expediting the data collection process in metamodel building as described in Chapter 4.

3.2 The Stochastic Fluid Model (SFM) Setting

The SFM setting is based on the fluid-flow worldview, where "liquid molecules" flow in a continuous fashion. The basic SFM, used in [80] and shown in Fig. 3.2, consists of a single-server (spigot) preceded by a buffer (fluid storage tank), and it is characterized by five stochastic processes, all defined on a common probability space $(\Omega, \mathcal{F}, P)$ as follows:

- $\{\alpha(t)\}$: the input flow (inflow) rate to the SFM,
- $\{\beta(t)\}$: the service rate, i.e., the maximal fluid discharge rate from the server,
- $\{\delta(t)\}$: the output flow (outflow) rate from the SFM, i.e., the actual fluid discharge rate from the server,
- $\{x(t)\}$: the buffer occupancy or buffer content, i.e., the volume of fluid in the buffer,
- $\{\gamma(t)\}$: the overflow (spillover) rate due to excessive incoming fluid at a full buffer.

The above processes evolve over a time interval $[0, T]$ for a given fixed $T > 0$. The inflow process $\{\alpha(t)\}$ and the service-rate process $\{\beta(t)\}$ are assumed to be right-continuous piecewise constant, with $0 \leq \alpha_{\min} \leq \alpha(t) \leq \alpha_{\max} < \infty$ and $0 \leq \beta_{\min} \leq \beta(t) \leq \beta_{\max} < \infty$. Let θ denote the size of the buffer, which is the variable parameter we will concentrate on for the purpose of IPA. The processes $\{\alpha(t)\}$ and $\{\beta(t)\}$, along with the buffer size θ, define the behavior of the SFM. In particular, they determine the buffer content, $x(\theta; t)$, the overflow rate $\gamma(\theta; t)$, and the output flow $\delta(\theta; t)$. The notational dependence on θ indicates that we will analyze performance metrics as functions of the given θ. We will assume that the real-valued parameter θ is confined to a closed and bounded (compact) interval Θ; to avoid unnecessary technical complications, we assume that $\theta > 0$ for all $\theta \in \Theta$.

The buffer content $x(\theta; t)$ is determined by the following one-sided differential equation,

$$\frac{dx(\theta;t)}{dt^+} = \begin{cases} 0, & \text{if } x(\theta;t) = 0 \text{ and } \alpha(t) - \beta(t) \leq 0, \\ 0, & \text{if } x(\theta;t) = \theta \text{ and } \alpha(t) - \beta(t) \geq 0, \\ \alpha(t) - \beta(t), & \text{otherwise} \end{cases} \tag{3.1}$$

with the initial condition $x(\theta;0) = x_0$ for some given x_0; for simplicity, we set $x_0 = 0$ throughout the chapter. The outflow rate $\delta(\theta;t)$ is given by

$$\delta(\theta;t) \;=\; \begin{cases} \beta(t), & \text{if } x(\theta;t) > 0, \\ \alpha(t), & \text{if } x(\theta;t) = 0, \end{cases} \tag{3.2}$$

where we point out that if we allow $\theta = 0$, then $\delta(\theta;t) \;= \min\{\alpha(t), \beta(t)\}$. The overflow rate $\gamma(\theta;t)$ is given by

$$\gamma(\theta;t) \;=\; \begin{cases} \max\{\alpha(t) - \beta(t), 0\}, & \text{if } x(\theta;t) = \theta, \\ 0, & \text{if } x(\theta;t) < \theta. \end{cases} \tag{3.3}$$

This SFM can be viewed as a dynamic system whose input consists of the two *defining* processes $\{\alpha(t)\}$ and $\{\beta(t)\}$ along with the buffer size θ, its state is comprised of the buffer content process, and its output includes the outflow and overflow processes. The state and output processes are referred to as *derived* processes, since they are determined by the defining processes. Since the input sample functions (realizations) of $\{\alpha(t)\}$ and $\{\beta(t)\}$ are piecewise constant and right-continuous, the state trajectory $x(\theta;t)$ is piecewise linear and continuous in t, and the output function $\gamma(\theta;t)$ is piecewise constant. Moreover, the state trajectory can be decomposed into two kinds of intervals: *empty periods* and *busy periods.* Empty Periods (EP) are maximal intervals during which the buffer is empty, while Busy Periods (BP) are supremal intervals during which the buffer is nonempty. Observe that during an EP the system is not necessarily idle since the server may be active; see (3.2). Note also that since $x(\theta;t)$ is continuous in t, EPs are always closed intervals, whereas BPs are open intervals unless containing one of the end points 0 or T. The outflow process $\{\delta(t)\}$ becomes important in modeling networks of SFMs and it will not concern us any further here, since our interest in this chapter lies in single-node systems.

Let $\mathcal{L}(\theta) : \Theta \to \mathbb{R}$ be a random function defined over the underlying probability space $(\Omega, \mathcal{F}, P)$. Strictly speaking, we write $\mathcal{L}(\theta, \omega)$ to indicate that this sample function depends on the sample point $\omega \in \Omega$, but will suppress ω unless it is necessary to stress this fact. In what follows, we will consider two performance metrics, the *Loss Volume* $L_T(\theta)$ and the *Cumulative Workload* (or just *Work*) $Q_T(\theta)$, both defined on the interval $[0, T]$ via the following equations:

$$L_T(\theta) \;=\; \int_0^T \gamma(\theta;t)dt, \tag{3.4}$$

$$Q_T(\theta) \;=\; \int_0^T x(\theta;t)dt, \tag{3.5}$$

where, as already mentioned, we assume that $x(\theta;0) = 0$. Observe that $\frac{1}{T}E\left[L_T(\theta)\right]$ is the *Expected Loss Rate* over the interval $[0, T]$, a common performance metric of interest (from which related metrics such as *Loss Probability* can also be derived). Similarly, $\frac{1}{T}E\left[Q_T(\theta)\right]$ is the *Expected Buffer Content* over $[0, T]$. In this chapter we are interested in estimates of $dJ_L(\theta)/d\theta$ and $dJ_Q(\theta)/d\theta$ provided by the sample derivatives $dL_T(\theta)/d\theta$ and $dQ_T(\theta)/d\theta$. Accordingly, the objective of the next section is the estimation of the derivatives of $J_L(\theta)$ and $J_Q(\theta)$, which we will pursue through Infinitesimal Perturbation Analysis (IPA) techniques [41, 20]). Henceforth we shall use the "prime" notation to denote derivatives with respect to

θ, and will proceed to estimate the derivatives $J_L^{'}(\theta)$ and $J_Q^{'}(\theta)$. The corresponding sample derivatives are denoted by $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$, respectively.

3.3 Infinitesimal Perturbation Analysis (IPA) with respect to Buffer Size or Threshold

We will concentrate on the buffer size θ in the SFM described above or, equivalently, a threshold parameter used for buffer control. We assume that the processes $\{\alpha(t)\}$ and $\{\beta(t)\}$ are independent of θ and of the buffer content. Thus, we consider network settings operating with protocols such as ATM and UDP, but not TCP. Our objective is to estimate the derivatives $J_L^{'}(\theta)$ and $J_Q^{'}(\theta)$ through the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ which are commonly referred to as Infinitesimal Perturbation Analysis (IPA) estimators; comprehensive discussions of IPA and its applications can be found in [41, 20]. The IPA derivative-estimation technique computes $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ along an observed sample path ω. An IPA-based estimate $\mathcal{L}'(\theta)$ of a performance metric derivative $dE[\mathcal{L}(\theta)]/d\theta$ is *unbiased* if $dE[\mathcal{L}(\theta)]/d\theta = E[\mathcal{L}'(\theta)]$. Unbiasedness is the principal condition for making the application of IPA practical, since it enables the use of the sample (IPA) derivative in control and optimization methods that employ stochastic gradient-based techniques.

We consider sample paths of the SFM over $[0, T]$. For a fixed $\theta \in \Theta$, the interval $[0, T]$ is divided into alternating EPs and BPs. Suppose there are K busy periods denoted by $\mathcal{B}_k$, $k = 1, \ldots, K$, in increasing order. Then, by (3.4)-(3.5), the sample performance functions assume the following form:

$$L_T(\theta) = \sum_{k=1}^{K} \int_{\mathcal{B}_k} \gamma(\theta; t) dt, \tag{3.6}$$

$$Q_T(\theta) = \sum_{k=1}^{K} \int_{\mathcal{B}_k} x(\theta; t) dt. \tag{3.7}$$

As mentioned earlier, the processes $\{\alpha(t)\}$ and $\{\beta(t)\}$ are assumed piecewise constant. This implies that, w.p.1, there exist a random integer $N(T) > 0$ and an increasing sequence of time points $0 = t_0 < t_1 < \ldots < t_{N(T)} < t_{N(T)+1} = T$, generally dependent upon the sample path ω, such that t_i is a jump (discontinuity) point of $\alpha(t) - \beta(t)$; clearly, $\alpha(t) - \beta(t)$ is continuous at all points other than $t_0, \ldots, t_{N(T)}$. We will assume that $N(T)$ has a finite expectation, i.e., $E[N(T)] < \infty$.

Viewed as a discrete-event system, an *event* in a sample path of the SFM may be either *exogenous* or *endogenous*. An exogenous event is a jump in either $\{\alpha(t)\}$ or $\{\beta(t)\}$. An endogenous event is defined to occur when the buffer becomes full or empty. We note that the times at which the buffer *ceases to be* full or empty are locally independent of θ, because they correspond to a change of sign in the difference function $\alpha(t) - \beta(t)$ (by a random function $f(\theta)$ being "locally independent" of θ we mean that for a given θ there exists $\Delta\theta > 0$ such that for every $\bar{\theta} \in (\theta - \Delta\theta, \theta + \Delta\theta)$, w.p.1 $f(\bar{\theta}) = f(\theta)$, where $\Delta\theta$ may depend on both θ and on the sample path). Thus, given a BP $\mathcal{B}_k$, its starting point is one

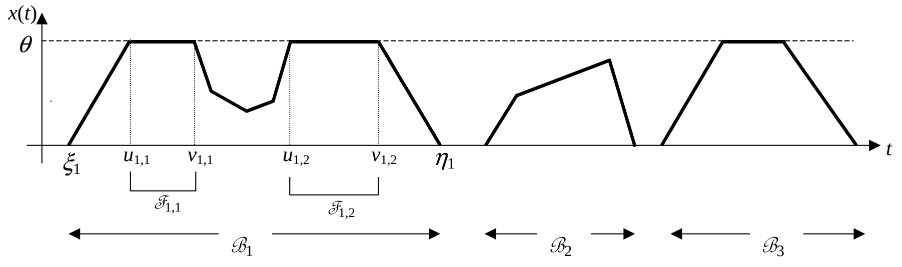


Figure 3.3: A typical sample path of a SFM

where the buffer ceases to be empty and is therefore locally independent of θ, while its end point generally depends on θ. Denoting these points by ξ_k and $\eta_k(\theta)$ we express $\mathcal{B}_k$ as

$$\mathcal{B}_k \; = \; (\xi_k, \eta_k(\theta)), \qquad k = 1, \ldots, K$$

for some random integer K. The BPs can be classified according to whether some overflow occurs during them or not. Thus, we define the random set

$$\begin{aligned}\Phi(\theta) \; := \; \{k \in \{1, \ldots, K\} : \; & x(t) = \theta, \\ & \alpha(t) - \beta(t) > 0 \text{ for some } t \in (\xi_k, \eta_k(\theta))\}.\end{aligned}$$

For every $k \in \Phi(\theta)$, there is a (random) number $M_k \geq 1$ of *overflow periods* in $\mathcal{B}_k$, i.e., intervals during which the buffer is full and $\alpha(t) - \beta(t) > 0$. Let us denote these overflow periods by $\mathcal{F}_{k,m}$, $m = 1, \ldots, M_k$, in increasing order and express them as $\mathcal{F}_{k,m} \; = \; [u_{k,m}(\theta), v_{k,m}]$, $k = 1, \ldots, K$. Observe that the starting time $u_{k,m}(\theta)$ generally depends on θ, whereas the ending time $v_{k,m}$ is locally independent of θ, since it corresponds to a change of sign in the difference function $\alpha(t) - \beta(t)$, which has been assumed independent of θ. Finally let

$$B(\theta) = |\Phi(\theta)| \tag{3.8}$$

where $|\cdot|$ denotes the cardinality of a set, i.e., $B(\theta)$ is the number of BPs in $[0, T]$ during which some overflow is observed. To summarize:

- There are K busy periods in $[0, T]$, with $\mathcal{B}_k = (\xi_k, \eta_k(\theta)), \;\; k = 1, \ldots, K$.
- $k \in \Phi(\theta)$ iff some overflow occurs during $\mathcal{B}_k$; we set $B(\theta) = |\Phi(\theta)|$.
- For each $k \in \Phi(\theta)$, there are M_k overflow periods in $\mathcal{B}_k$, i.e., $\mathcal{F}_{k,m} = [u_{k,m}(\theta), v_{k,m}]$, $m = 1, \ldots, M_k$.

A typical sample path is shown in Fig. 3.3, where $K = 3$, $\Phi = \{1, 3\}$, $M_1 = 2$, $M_2 = 0$, $M_3 = 1$.

Next we present the IPA derivative estimates. Note that the results of the next section can be derived using either finite differences or direct sample differentiation. We only present the main results without any proofs. The interested reader is referred to Appendix B or [17] for details.

3.3.1 Infinitesimal Perturbation Analysis

In this subsection, we derive explicitly the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ of the loss volume and work, defined in (3.6) and (3.7), respectively. We then show that they provide unbiased estimators of the expected loss volume sensitivity $dE[L_T(\theta)]/d\theta$ and the expected work sensitivity $dE[Q_T(\theta)]/d\theta$.

Since we are concerned with the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$, we have to identify conditions under which they exist. Observe that any endogenous event time (a time point when the buffer becomes full or empty) is generally a function of θ; see also (3.1). Denoting this point by $t(\theta)$, the derivative $t'(\theta)$ exists as long as $t(\theta)$ is not a jump point of the difference process $\{\alpha(t) - \beta(t)\}$. Recall that the times at which the buffer ceases to be full or empty are locally independent of θ, because they correspond to a change-of-sign of the difference sample function $\alpha(t)-\beta(t)$, which does not depend on θ. Excluding the possibility of the simultaneous occurrence of two events, the only situation preventing the existence of the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ involves an interval during which $x(t) = \theta$ and $\alpha(t) - \beta(t) = 0$, as seen in (3.3)); in this case, the one-sided derivatives of $L_T(\theta)$and $Q_T(\theta)$ exist and can be obtained with the approach of the previous section. In order to keep the analysis simple, we focus only on the differentiable case. Therefore, the analysis that follows rests on the following technical conditions:

Assumption 1
a. *W.p.1, $\alpha(t) - \beta(t) \neq 0$.*
b. *For every $\theta \in \Theta$, w.p.1, no two events may occur at the same time.*

Remark 1 *We stress the fact that the above conditions for ensuring the existence of the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ are very mild. Part* b *above is satisfied whenever the cdf's (or conditional cdf's) characterizing the intervals between exogenous event occurrences are continuous. For example, in the simple case where $\beta(t) = \beta$ and $\alpha(t)$ can only take two values, 0 and $\alpha > \beta$, suppose that the inflow process switches from α to 0 after $\theta/(\alpha - \beta)$ time units w.p. 1. The buffer then becomes full exactly when an exogenous event occurs, and the loss volume sample function experiences a discontinuity w.p. 1. Such situations can only arise for a small finite subset of Θ (for which one can still calculate either the left or right derivatives) and they are of limited practical consequence.*

We next derive the IPA derivatives of $L_T(\theta)$and $Q_T(\theta)$. Recall that $B(\theta) = |\Phi(\theta)|$, i.e., the number of BPs containing at least one overflow period.

Theorem 1 *For every $\theta \in \Theta$,*
$$L_T^{'}(\theta) \;=\; -B(\theta). \tag{3.9}$$

Theorem 2 *For every $\theta \in \Theta$,*
$$Q_T^{'}(\theta) \;=\; \sum_{k\in\Phi(\theta)} [\eta_k(\theta) - u_{k,1}(\theta)]. \tag{3.10}$$

In simple terms, the contribution of a BP, $\mathcal{B}_k$, to the sample derivative $Q_T^{'}(\theta)$ in (3.10) is the length of the interval defined by the first point at which the buffer becomes full and the end of the BP. Once again, as in (3.9), observe that the IPA derivative $Q_T^{'}(\theta)$ is nonparametric, since it requires only the recording of times at which the buffer becomes full (i.e., $u_{k,1}(\theta)$) and empty (i.e., $\eta_k(\theta)$) for any $\mathcal{B}_k$ with $k \in \Phi(\theta)$.

IPA Unbiasedness

We next show the unbiasedness of the IPA derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ obtained above. In general, the unbiasedness of an IPA derivative $\mathcal{L}'(\theta)$ has been shown to be ensured by the following two conditions (see [71], Lemma A2, p.70):

Condition 1. For every $\theta \in \Theta$, the sample derivative $\mathcal{L}'(\theta)$ exists w.p.1.

Condition 2. W.p.1, the random function $\mathcal{L}(\theta)$ is Lipschitz continuous throughout Θ, and the (generally random) Lipschitz constant has a finite first moment.

Consequently, establishing the unbiasedness of $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ as estimators of $dE[L_T(\theta)]/d\theta$ and $dE[Q_T(\theta)]/d\theta$, respectively, reduces to verifying the Lipschitz continuity of $L_T(\theta)$ and $Q_T(\theta)$ with appropriate Lipschitz constants. Recall that $N(T)$ is the random number of all exogenous events in $[0, T]$ and that we have assumed $E[N(T)] < \infty$.

Theorem 3 *Under **Assumption 1**,*

1. *If $E[N(T)] < \infty$, then the IPA derivative $L_T^{'}(\theta)$ is an unbiased estimator of $dE[L_T(\theta)]/d\theta$.*
2. *The IPA derivative $Q_T^{'}(\theta)$ is an unbiased estimator of $dE[Q_T(\theta)]/d\theta$.*

Remark. For the more commonly used performance metrics $\frac{1}{T}E\left[L_T(\theta)\right]$ (the Expected Loss Rate over $[0, T]$) and $\frac{1}{T}E\left[Q_T(\theta)\right]$ (the Expected Buffer Content over $[0, T]$), the Lipschitz constants in Theorem 3 become $N(T)/T$ and 1, respectively. As $T \to \infty$, the former quantity typically converges to the exogenous event rate.

3.3.2 IPA Estimation Algorithm

Algorithm 1

```
• Initialize a counter $\mathcal{C} := 0$ and a cumulative timer $\mathcal{T} := 0$.
• Initialize $\tau := 0$.
• If an overflow event is observed at time $t$ and $\tau = 0$:
    – Set $\tau := t$
• If a busy period ends at time $t$ and $\tau > 0$:
    – Set $\mathcal{C} := \mathcal{C} - 1$ and $\mathcal{T} := \mathcal{T} + (t - \tau)$
    – Reset $\tau := 0$.
• If $t = T$, and $\tau > 0$:
```

– *Set* $\mathcal{C} := \mathcal{C} - 1$ *and* $\mathcal{T} := \mathcal{T} + (t - \tau)$.

The final values of $\mathcal{C}$ and $\mathcal{T}$ provide the IPA derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ respectively. We remark that the "overflow" and "end of BP" events are readily observable during actual network operation. In addition, we point out once again that these estimates are independent of all underlying stochastic features, including traffic and processing rates. Finally, the algorithm is easily modified to apply to any interval $[T_1, T_2]$.

3.4 Conclusions and Future Work

Stochastic Fluid Models (SFM) can adequately describe the dynamics of complex discrete event systems or constitute primary models in their own right. In this chapter, we have considered single-node SFMs from the standpoint of IPA derivative estimation. In particular, we have developed IPA estimators for the loss volume and work as functions of the buffer size, and shown them to be unbiased and nonparametric. The simplicity of the estimators and their nonparametric property suggest their application to on-line optimization problems.

The sample derivative analysis holds the promise of considerable extensions to multiple SFMs as models of actual networks and to multiple flow classes that can be used for differentiating traffic classes with different Quality-of-Service (QoS) requirements. Ongoing research has already led to very encouraging results, reported in [17, 16, 15], involving IPA estimators and associated optimization for flow control purposes in multi-node models.

Chapter 4

NEURAL NETWORK METAMODELING

Simulation is one of the most powerful tools for modeling and evaluating the performance of complex systems, however, it is computationally slow. One approach to overcome this limitation is to develop a "metamodel". In other words, generate a "surrogate" model of the original system that accurately captures the relationships between input and output, yet it is computationally more efficient than simulation. Neural networks (NN) are known to be good function approximators and thus make good metamodel candidates. During training, a NN is presented with several input/output pairs, and is expected to learn the functional relationship between inputs and outputs of the simulation model. So, a trained net can *predict* the output for inputs other than the ones presented during training. This ability of NNs to generalize depends on the number of training pairs used. In general, a large number of such pairs is required and, since they are obtained through simulation, the metamodel development is slow. When using concurrent simulation, as in Chapter 3, we can obtain sensitivity information with respect to various input parameters. In this chapter, we investigate the use of sensitivity information to reduce the simulation effort required for training a NN metamodel.

4.1 Introduction

Simulation is arguably the most versatile and general-purpose tool available today for modeling complex systems such as Discrete Event Systems (DES). It can be used for performance evaluation, system design, decision making, and planning. Such applications typically involve the use of simulation to answer a multitude of "what-if" questions under various scenarios, each corresponding to different parameters, designs or decisions. However, simulation is notoriously time consuming. For complex systems, performance evaluation under a single set of input parameters can take several minutes even hours. As a result, it is impractical (if at all feasible) to perform any parametric study of system performance, especially for systems with a large parameter space. Unless substantial speedup of the performance evaluation process can be achieved, systematic performance studies of most real-world problems are beyond reach, even with supercomputers.

One alternative for achieving the required speedup is through "metamodeling". In this framework, any simulator is viewed as a function that maps any vector of input parameters $\mathbf{x} = [x_1, \cdots, x_N]$ to a set of M performance metrics of interest $\mathbf{y} = [y_1, \cdots, y_M]$, that is

$$\mathbf{y} = \Phi(\mathbf{x}). \tag{4.1}$$

Since the evaluation of the function $\Phi(\cdot)$ is generally complex and time consuming, metamodeling seeks a much simpler and computationally more efficient "surrogate model" $\hat{\Phi}(\cdot)$ such that

$$\hat{\Phi}(\mathbf{x}) \approx \Phi(\mathbf{x}). \tag{4.2}$$

for all input parameters of interest. A typical approach for building $\hat{\Phi}(\cdot)$ is to use simulation to obtain a training set of Q input-output pairs $\{(\mathbf{x}_i, \mathbf{y}_i),\ i = 1, \cdots, Q\}$, and try to determine a function that captures the input-output relationship that generated the Q samples. Of course, the expectation is that the surrogate model will be such that (4.2) will hold not only for the training pairs, but also for *any* $\mathbf{x}$ in some domain of interest. This ability of the surrogate model to produce a reasonable response to an input that is not included in the training set is referred to as *generalization*.

Several authors have addressed simulation metamodeling. [88] used polynomial fitting to develop a metamodel for a Tactical Electronic Reconnaissance Simulation Model (TERSM) that estimates the number of ground-based radar sites detected by a reconnaissance aircraft as a function of its flying mission. Other approaches have used statistical analysis. [70] used least squares estimation for non-linear metamodel estimation while [24] used regression using Bayesian methods. Neural networks (NN) are generally known as good function approximators and thus make good candidates for surrogate functions. [45] have used a backpropagation neural net to capture the behavior of a Command and Control (C2) network. In some of our earlier work [14], we used a Cascade Correlation NN [26] to generate metamodels for the TERSM mentioned above and for an Aircraft Refueling and Maintenance System (ARMS).

Neural network metamodels, though versatile, generally require a large number of training pairs before they acquire good generalization capabilities. This implies that many simulation runs are necessary to build a metamodel. To address this problem, in our earlier work [14] we also proposed the use of *Concurrent Estimation* [21] as a possible way of collecting more training pairs from a single simulation run. In this work we propose a different approach where we use sensitivity (derivative) information to train the neural network.

More specifically, it is by now well-documented in the literature that the nature of sample paths of Discrete Event Systems (DES) can be exploited so as to extract a significant amount of information, beyond merely an estimate of a performance measure $\Phi(\mathbf{x})$. It has been shown that observing a sample path under some parameter value x allows us to efficiently obtain estimates of derivatives of the form $d\Phi/dx$ which are in many cases unbiased and strongly consistent (e.g., see [11, 31, 41] where Infinitesimal Perturbation Analysis (IPA) and its extensions are described). The question that arises then is how the sensitivity information, made available by PA, can be used to construct metamodels. In this chapter we recognize that since (4.2) must hold for all $\mathbf{x}$ in the domain of interest, the partial derivatives of the two functions with respect to any x_i should also be equal. As a result, we modify the standard backpropagation algorithm to also use this information as explained in Section 4.4.

To our knowledge, this approach of using sensitivity information to reduce the training sample size is new. For the "classification" problem (as opposed to the "function approximation" problem we investigate in this chapter) several authors have addressed the issue of determining the minimum training sample size for a neural network to have good generalization properties. [7] found that for a network with N nodes and W weights, the number of randomly selected samples required to achieve correct classification for at least $(1 - \frac{\epsilon}{2})$ fraction of the test examples is $m \geq O\left(\frac{W}{\epsilon} \log \frac{N}{\epsilon}\right)$. [59] found a tighter bound assuming that the training samples are chosen close to the cluster boundaries. The generalization performance of practical algorithms is also the focal point in [75]. These authors use the so called "ill-disposed" algorithm to derive a probability distribution that allows them to determine a more realistic bound on the sample size as well as the average generalization error.

This chapter is organized as follows. In the next section we present the notation that we will use in the sequel. In Section 4.3 we briefly describe the standard backpropagation neural network and in Section 4.4 we modify the algorithm to include any available sensitivity information. In Section 4.5 we demonstrate the potential advantages of the approach with two numerical examples. Finally we close with conclusions and future plans in Section 4.6.

4.2 Notation

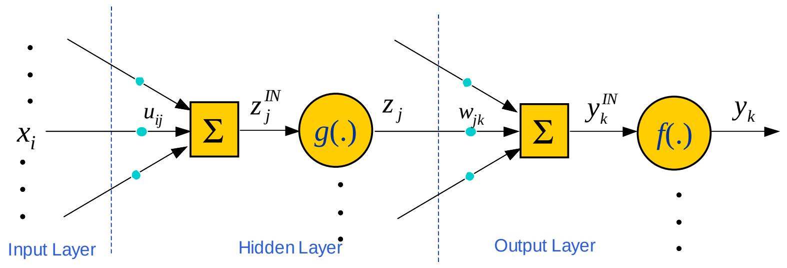


Figure 4.1: 3-Layer neural network

For the purposes of this chapter, we assume a 3-layer neural network (see Fig. 4.1) where

M**:** Number of units (neurons) in the output layer.

H**:** Number of units in the hidden layer.

N**:** Number of units in the input layer.

y_k**:** Output of the kth unit of the output layer, $k = 1, \cdots, M$.

z_j**:** Output of jth hidden unit, $j = 0, \cdots, H$ ($z_0 = 1$ corresponds to the bias input of output layer units).

x_i**:** Input of the ith unit, $i = 0, \cdots, N$ ($x_0 = 1$ corresponds to the bias input of hidden layer units).

$\mathbf{t}_p$ $= [t_k]_p$. Target output given an input vector $\mathbf{x}_p$, where $k = 1, \cdots, M$, $p = 1, \cdots, Q$ and Q is the number of training pairs.

d_{ki}**:** Sensitivity of kth network output with respect to its ith input, $d_{ki} = \frac{\partial y_k}{\partial x_i}$.

$\mathbf{S}_p$ $= [s_{ki}]_p$ Target sensitivity (d_{ki}) given an input vector $\mathbf{x}_p$. (For DES, we assume that this information is obtained through some Perturbation Analysis (PA) technique).

$f(\cdot)$**:** Activation function of output layer units.

$g(\cdot)$**:** Activation function of hidden layer units.

4.3 Backpropagation Neural Net

The activation of the kth output unit of a standard, three layer backpropagation neural network (BPNN) as a function of the input $\mathbf{x} = [x_1, \cdots, x_n]$ is given by:

$$\begin{aligned} y_k &= f\left(y_k^{IN}\right) && (4.3) \\ y_k^{IN} &= \sum_{j=0}^{H} w_{jk} z_j && (4.4) \\ z_j &= g\left(z_j^{IN}\right) && (4.5) \\ z_j^{IN} &= \sum_{i=0}^{N} u_{ij} x_i && (4.6) \end{aligned}$$

where, w_{jk} is the weight from the jth hidden unit to the input of the kth output unit and u_{ij} is the weight from the ith input to the jth hidden unit.

The learning procedure of the backpropagation neural network is based on minimizing the sum of squared errors. That is, minimize an error function of the form:

$$E = \frac{1}{2} \sum_{k=1}^{M} (t_k - y_k)^2. \tag{4.7}$$

The minimization is done by gradient descent methods, where backpropagation involves the chain rule to back propagate errors from the network's outputs to each of the network's weights, see [27] for details. Next, we investigate a possible way of utilizing the sensitivity information that can be obtained through some PA technique.

4.4 Derivative Backpropagation Neural Networks

If equation (4.2) is supposed to hold for all $\mathbf{x}$ in the domain of interest of $\mathbf{x}$, then it is reasonable to require that:

$$\frac{\partial \hat{\Phi}(\mathbf{x})}{\partial x_i} \approx \frac{\partial \Phi(\mathbf{x})}{\partial x_i}, \quad i = 1, \cdots, N. \tag{4.8}$$

In other words, the sensitivity of the neural net output with respect to each one of its inputs should be approximately equal to the sensitivity of the simulation model with respect to the same inputs. Though complex, it is possible to determine the neural network's sensitivity with respect to its inputs using calculus. Also, for several discrete event systems, the model's sensitivity with respect to its input parameters can be calculated using some perturbation analysis technique. Thus, the main idea behind our approach is to adapt (4.7) to account not only for the error in the output value, but for the error in the sensitivity as well. Therefore, the neural net training objective function becomes

$$E = \frac{\alpha}{2} \sum_{k=1}^{M} (t_k - y_k)^2 + \frac{\beta}{2} \sum_{k=1}^{M} \sum_{i=1}^{N} (s_{ki} - d_{ki})^2 . \tag{4.9}$$

The first term is the usual error term used in standard backpropagation neural networks. The second term, is the error in the sensitivity of the neural net compared to the sensitivity of the model. Finally, $0 \leq \alpha \leq 1$ and $\beta = 1 - \alpha$ are weighting factors that determine the importance to be associated to the derivative error. Note that if $\beta = 0$, then we get the standard backpropagation algorithm.

Next, if we are interested in minimizing the error function of (4.9), we need to determine the neural network's sensitivity with respect to its inputs, d_{ki}. For the 3-layer network we consider in this chapter, this is done by the chain rule of differentiation as shown below:

$$\begin{aligned} d_{ki} = \frac{\partial y_k}{\partial x_i} &= f'\left(y_k^{IN}\right) \frac{\partial y_k^{IN}}{\partial x_i} \\ &= f'\left(y_k^{IN}\right) \sum_{j=1}^{H} w_{jk} \frac{\partial z_j}{\partial x_i} \\ &= f'\left(y_k^{IN}\right) \sum_{j=1}^{H} w_{jk} g'\left(z_j^{IN}\right) \frac{\partial z_j^{IN}}{\partial x_i} \\ &= f'\left(y_k^{IN}\right) \sum_{j=1}^{H} w_{jk} g'\left(z_j^{IN}\right) u_{ij} \end{aligned} \tag{4.10}$$

Subsequently, if we want to use gradient based techniques to minimize the error function of (4.9) we need $\frac{\partial E}{\partial w_{jk}}$ and $\frac{\partial E}{\partial u_{ij}}$ for all i, j, k which are derived next, through repetitive use of the chain rule of differentiation.

$$\begin{aligned}
\frac{\partial E}{\partial w_{JK}} &= -\alpha(t_K - y_K)\frac{\partial y_k}{\partial w_{JK}} - \beta \sum_{i=1}^{N}\left[e_{Ki}\frac{\partial d_{Ki}}{\partial w_{JK}}\right] \\
&= -\alpha(t_K - y_K) f'\left(y_K^{IN}\right) z_J \\
&\quad -\beta\sum_{i=1}^{N}\left[e_{Ki}\left(f''\left(y_K^{IN}\right) z_J S(K,i) + f'\left(y_K^{IN}\right) g'\left(z_J^{IN}\right) u_{iJ}\right)\right] \qquad (4.11)
\end{aligned}$$

where

$$e_{ki} = (s_{ki} - d_{ki})$$

and

$$S(k,i) = \sum_{j=1}^{H} w_{jk} g'\left(z_j^{IN}\right) u_{ij}.$$

Similarly,

$$\begin{aligned}
\frac{\partial E}{\partial u_{IJ}} &= -\alpha\sum_{k=1}^{M}(t_k - y_k) f'\left(y_k^{IN}\right) w_{Jk} g'\left(z_J^{IN}\right) x_I \\
&\quad -\beta\sum_{k=1}^{M}\sum_{i=1}^{N}\left[e_{ki}\left(f''\left(y_k^{IN}\right) w_{Jk} g'\left(z_J^{IN}\right) x_I S(k,i)\right.\right. \\
&\quad \left.\left. +\ f'\left(y_k^{IN}\right) w_{Jk}\left(g''\left(z_J^{IN}\right) u_{iJ} x_I + g'\left(z_J^{IN}\right)\right)\right)\right] \qquad (4.12)
\end{aligned}$$

Note that these expressions get considerably simpler when the activation function of the output layer units are linear. In this case, $f(x) = x$, $f'(x) = 1$, and $f''(x) = 0$. Therefore,

$$\frac{\partial E}{\partial w_{JK}} = -\alpha(t_K - y_K) z_J - \beta g'\left(z_J^{IN}\right)\sum_{i=1}^{N}(s_{Ki} - d_{Ki}) u_{iJ} \qquad (4.13)$$

and

$$\begin{aligned}
\frac{\partial E}{\partial u_{IJ}} &= -\alpha g'\left(z_J^{IN}\right) x_I \sum_{k=1}^{M}(t_k - y_k) w_{Jk} \\
&\quad -\beta\sum_{k=1}^{M}\sum_{i=1}^{N} e_{ki} w_{Jk}\cdot\left(g''\left(z_J^{IN}\right) u_{iJ} x_I + g'\left(z_J^{IN}\right)\right) \qquad (4.14)
\end{aligned}$$

Finally, the weight updates at every iteration t are given by

$$w_{jk}(t+1) = w_{jk}(t) - \gamma\frac{\partial E}{\partial w_{jk}} \qquad (4.15)$$

and

$$u_{ij}(t+1) = u_{ij}(t) - \gamma\frac{\partial E}{\partial u_{ij}} \qquad (4.16)$$

for all $i = 0, \cdots, N$, $j = 0, \cdots, H$, $k = 1, \cdots, M$. Where γ is the learning rate and $\frac{\partial E}{\partial w_{jk}}$ and $\frac{\partial E}{\partial u_{ij}}$ are given by (4.11) and (4.12) respectively (Note that $i = 0$ corresponds to the bias input of a neuron). In the sequel, this will be referred to as the Derivative Backpropagation Neural Network (DBPNN). At this point, it is worth pointing out that apart from the learning rate γ one needs to determine the weight to be given to the derivative error $\beta = 1 - \alpha$. This is an important factor that may affect the convergence of the algorithm as discussed later in the chapter.

4.5 Numerical Results

In this section we present some results that show the benefit of using the DBPNN training algorithm described in equations (4.15) and (4.16).

In our first experiment we try to approximate the function $y = x^2$ in the interval $[-10, 10]$. For this experiment, we use a neural network with 20 hidden units. First, we used just three input-output pairs $\{(-10, 100), (0, 0), (10, 100)\}$ to train a standard back-propagation neural network (BPNN). Subsequently, we added the derivatives at these three points and used the information to train a network using DBPNN. For this experiment we used $\beta = 0.1$. The outputs of the two networks as well as the target output function are shown in Fig. 4.2. As seen in the figure, the DBPNN approximates the target function much better than the standard BPNN. Also shown in the figure is the absolute value of the approximation error of each network for every value of x in $[-10, 10]$ which also demonstrates the benefit of DBPNN.

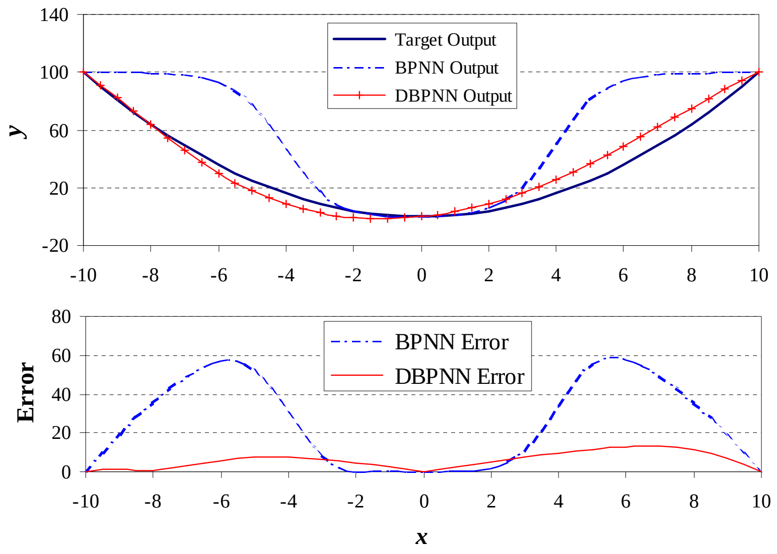


Figure 4.2: Approximation of the function $y = x^2$ using NNs

In order to compare the generalization ability of each network, we integrate the area under the error curve of each network and plot it in Fig. 4.3 as a function of the number of points used during the training of the two networks. As seen in the figure, DBPNN achieves much better generalization than standard backpropagation neural net, especially when the number of training points is small.

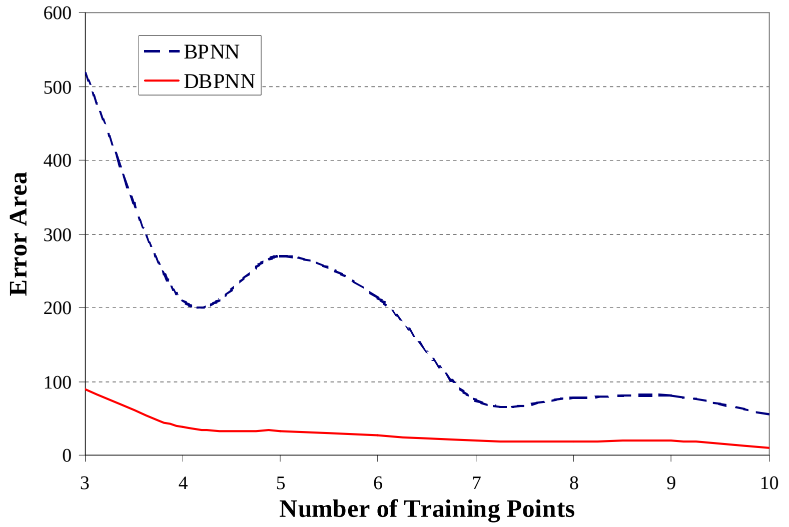


Figure 4.3: Area under the absolute error of the NN approximations for $y = x^2$

Next we consider an $M/M/1$ queueing network where we are interested in the average time that customers spend in the system S, as a function of the traffic intensity ρ. For this system, the IPA algorithm for determining $\frac{dS}{d\rho}$ is given in [11]. Fig. 4.4 shows the approximations generated by BPNN and DBPNN when both networks have 20 hidden units and are trained with only 5 train points and for DBPNN $\beta = 0.01$. As seen in the figure, DBPNN again achieves a much better generalization than the standard backpropagation network.

Finally, Fig. 4.5 shows that area under the generalization error for the two networks. Again, DBPNN achieves a much better generalization than BPNN especially for a small number of training points.

Note that for the second experiment we have set the β parameter to a very small value ($\beta = 0.01$). The reason is that as ρ approaches 1, the system time goes asymptotically to infinity and therefore the derivative at this point becomes very large. As a result, the derivative error for this point dominates the entire error function, so, the neural network in its effort to minimize the total error, it approximates *only* that derivative well, but does not approximate well the remaining points. Furthermore, we point out that other factors may also play a role in the value of the β parameter. For example, if we have noisy estimates of the derivatives, it may be preferable to also set β to a small value in order to avoid noise from taking over the output of the neural network. More research is required to determined how β is set.

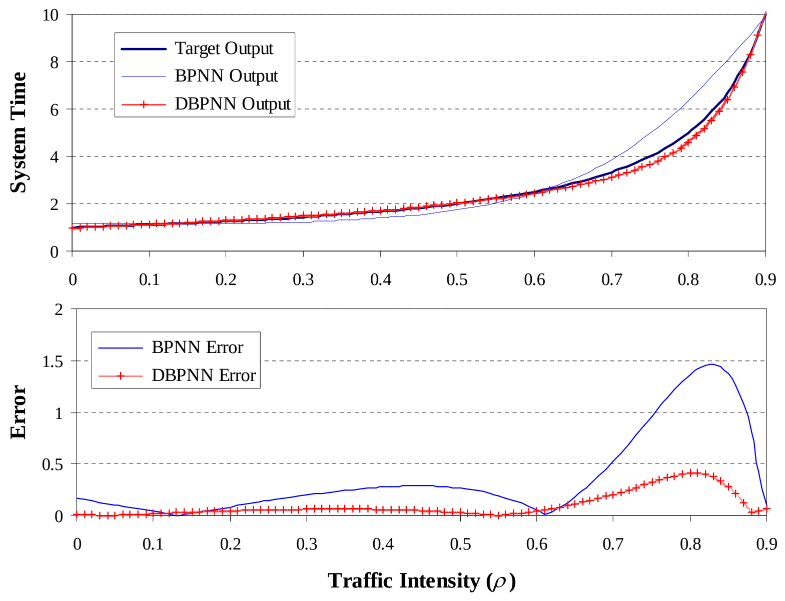


Figure 4.4: Approximation of the average system time in an $M/M/1$ queueing system

4.6 Summary

When dealing with complex systems, simulation is usually the only alternative for performance evaluation however, it is notoriously slow, thus the need for “metamodels”. Neural networks are considered as good function approximators thus make good metamodel candidates. However, if a neural net is to adequately learn the functional relationship between the inputs and outputs of a simulation model it requires a significant number of input/output pairs. Since such information can only be obtained through simulation, it means that the training phase of the neural network will be long. In this chapter, we investigate the use of sensitivity information in the training of backpropagation neural network. Some preliminary results indicate that the use of sensitivity information can significantly reduce the number of training input/output pairs required which in turn implies that the metamodel development phase will be expedited.

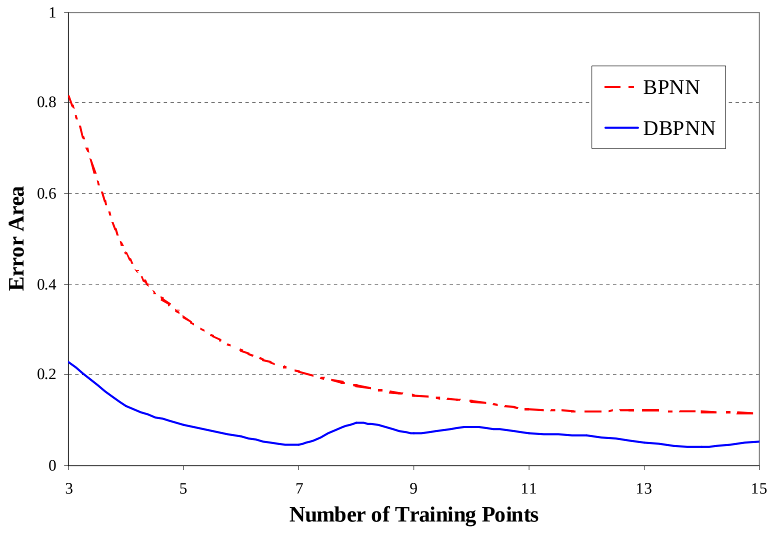


Figure 4.5: Area under the absolute error of the NN approximations for the average system time in an $M/M/1$ System

Chapter 5

HIERARCHICAL DECOMPOSITIONS AND THE CLUSTERING APPROACH

In this chapter, we discuss the use of clustering methods in hierarchical simulation of complex systems and present an application to a computer security problem. First, we discuss the basic concepts for multi-resolution simulation modelling of complex stochastic systems. We argue that high-resolution output data should be classified into groups that match underlying patterns or features of the system behavior before sending group averages to the low-resolution modules to keep the statistics fidelity. We propose high-dimensional data clustering as a key interfacing component between simulation modules with different resolutions and use unsupervised learning schemes to recover the patterns for the high-resolution simulation results. Subsequently, we give the examples of using Hidden Markov Model as an effective clustering tool for this task. Next, we apply the clustering techniques in the context of computer security and give some examples of using Hidden Markov Models (HMMs) for the purpose of system modelling for anomaly detection.

5.1 Introduction

In modelling complex systems it is impossible to mimic every detail through simulation. The common approach is to divide the whole system hierarchically into simpler modules, each with different simulation resolution. In this context, the output of a module becomes the input parameters to another, as illustrated in Fig. 5.1. The decomposed modules can be high-resolution or low-resolution models. High-resolution, e.g. the usual discrete-event simulation models, take detailed account for all possible events, but are generally time consuming. Low-resolution modules perform aggregate evaluation on the module functionality, i.e., determine what would happen "on the average". Such modules are less time consuming and can be any of the following components: differential equations, standard discrete-event simulation, and fluid simulation. Furthermore, the decomposed modules can also be an optimization or decision supporting tool.

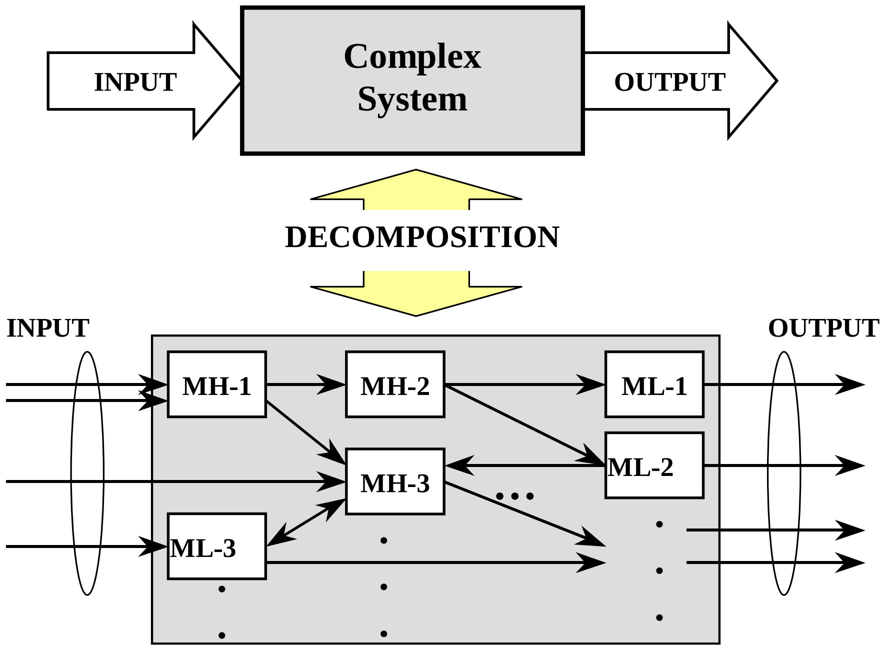


Figure 5.1: Decomposition of complex systems

In a hierarchical setting, the lower level simulator (typically with a high resolution) generates output data which are then taken as input for the higher level simulator (typically a low-resolution model). Hierarchical simulation is a common practice, but the design of hierarchy is always ad hoc. A popular practice is to use the mean values of variables from the lower level output as the input to the higher level. This implies that significant statistical information can be lost in this process, resulting in potentially completely inaccurate results. Especially when the ultimate output of the simulation process is of the form 0 or 1 (e.g.,“lose” or “win” a combat), such errors can provide the exact opposite of the real output.

Quite often, the system being simulated is such that the high-resolution model produces so widely divergent outputs that it does not make sense to summarize such output through a single average over the entire sample space. In such cases, we must subdivide the sample space into segments, and get the high- resolution model to produce an appropriate input to the low-resolution model for each such segment. Essentially, the low-resolution model will be broken down into a number of distinct components, one for each segment of the sample space. To carry out such a segmentation, the high-resolution paths first need to be grouped by their common features. These features then determine and feed the corresponding low-resolution model. This grouping procedure is essentially an unsupervised classification procedure, based on some similarity measure. This inspired us to use high-dimensional clustering techniques to group the high-resolution sample paths into meaningful clusters and pass on to the lower resolution modules with the statistical fidelity preserved.

5.1.1 Design of the Interface

A systematic design and analysis framework for multi-resolution complex systems is definitely needed. Here we present some fundamental components of such a framework. Our effort is directed at developing an interface between two simulation levels to preserve statistical fidelity to the maximum extent that available computing power allows. In a typical hierarchical simulation model, the lower lever consist of a high-resolution model, such as the discrete event simulator that generates several sample paths given some input parameters $\mathbf{u}$. The output of such simulation models is then used as input to the higher level model (typically a low-resolution model). The question is how much and what information we need to pass from the high-resolution to the low-resolution model such that statistical fidelity is preserved.

Note that each sample path generated by the high-resolution model is also a function of some randomness ω (a random number sequence generated through some random seed). Thus, any function evaluated over an observed sample path (e.g., $h(\mathbf{u}, \omega)$ is also a random variable. Typically, we are not interested in the value of $h(\mathbf{u}, \omega)$ obtained from a single sample path but rather the expectation $E\{h(\mathbf{u}, \omega)\}$. Based on this, in hierarchical simulation it is customary to use $E\{h(\mathbf{u}, \omega)\}$ as an input parameter to the higher level model as seen in Fig. 5.2. This is often highly unsatisfactory, since the mean often obscures important features of the high-resolution output. Said in another way, we are seeking $E\{L(h(\mathbf{u}, \omega))\}$, where $L(\cdot)$ is a function corresponding to the low-resolution model, but what we end up evaluating by passing a single average is $L(E\{h(\mathbf{u}, \omega)\})$; however, in general $E\{L(h(\mathbf{u}, \omega))\} \neq L(E\{h(\mathbf{u}, \omega)\})$.

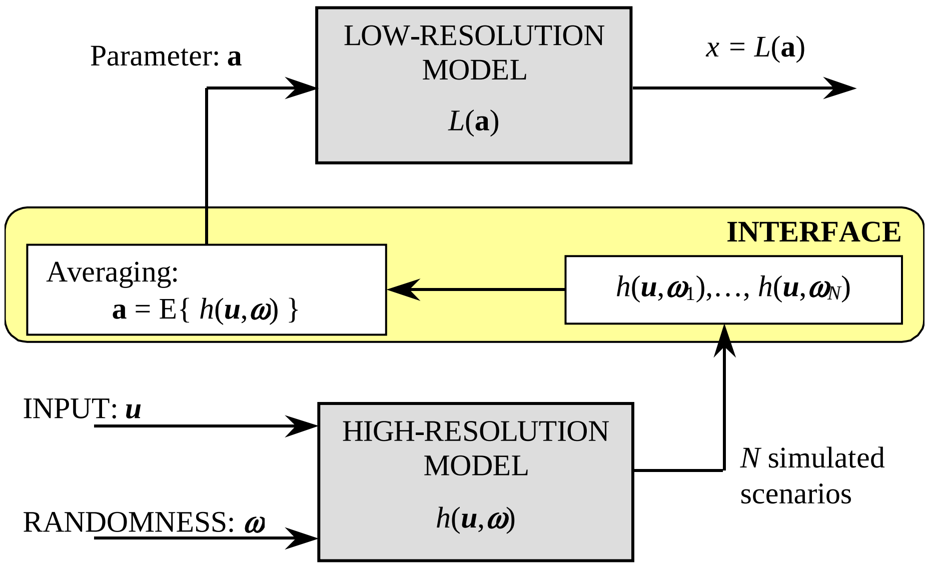
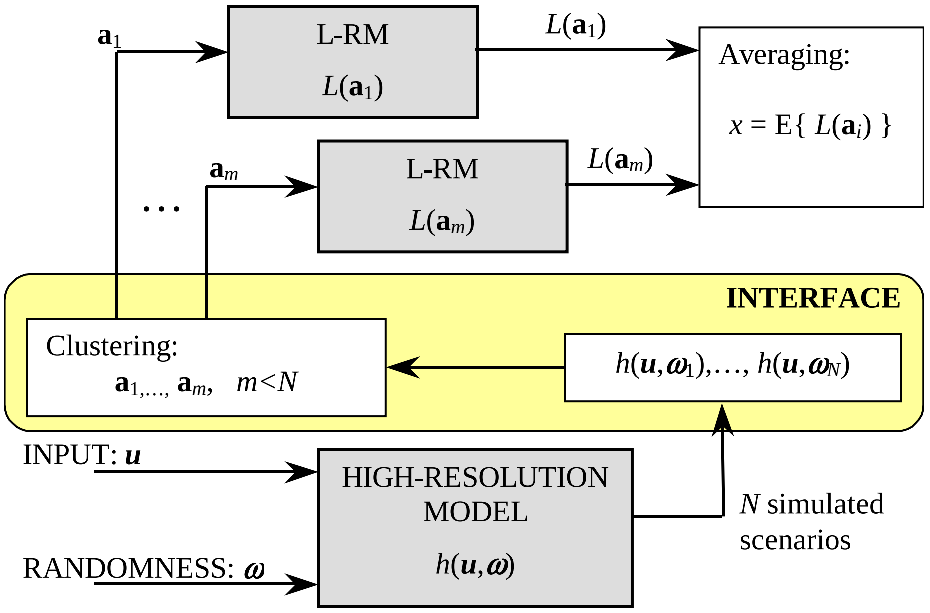


Figure 5.2: Hierarchical model interface: passing a simple average to the lower resolution model

To solve this problem we propose the use of clustering to identify groups of sample paths that have some "common features", and therefore, when averaged together do not cause the loss of too much information. This approach in shown in Fig. 5.3. From the N observed sample paths we identify $m < N$ groups that share some common features and determine m input parameters $\mathbf{a}_1, \ldots, \mathbf{a}_m$ where $\mathbf{a}_i = E\{h^i(\mathbf{u}, \omega)\}$ and $h^i(\cdot)$ identifies all sample paths in cluster i. Subsequently, each parameter $\mathbf{a}_i$ is used as an input to a lower resolution model and finally we obtain $E\{L(\mathbf{a}_i)\}$ over m low-resolution components, which we claim is a better estimate of the overall system output than the one obtained using a single average.



$\mathbf{a}_1$
L-RM
$L(\mathbf{a}_1)$
$L(\mathbf{a}_1)$
Averaging:
$x = \mathrm{E}\{\ L(\mathbf{a}_i)\ \}$
$\mathbf{a}_m$
...
L-RM
$L(\mathbf{a}_m)$
$L(\mathbf{a}_m)$
INTERFACE
Clustering:
$\mathbf{a}_{1,\ldots,}\ \mathbf{a}_m,\quad m<N$
$h(\boldsymbol{u},\boldsymbol{\omega}_1),\ldots,\ h(\boldsymbol{u},\boldsymbol{\omega}_N)$
INPUT: $\boldsymbol{u}$
HIGH-RESOLUTION MODEL
$h(\boldsymbol{u},\boldsymbol{\omega})$
RANDOMNESS: $\boldsymbol{\omega}$
N simulated scenarios

Figure 5.3: Hierarchical model interface: passing several averages to the lower resolution model, one for each cluster

One may pose the following question: Since the desired output is of the form $E\{L(h(\mathbf{u}, \omega))\}$, why bother with clustering at all when we can evaluate $L(E\{h(\mathbf{u}, \omega)\})$ for all N obtained samples and then perform the required expectation, especially since the low-resolution models are generally easy to evaluate? The answer to this question lies in the derivation of the low-resolution model. Typically, $L(\mathbf{a})$ assumes that $\mathbf{a}$ is an expectation and therefore it would be meaningless to use some quantity obtained from a single sample path.

5.1.2 Clustering Tools

Based on the above design principles of interface for the multi-resolution simulation framework, we have proposed two different types of clustering tools, i.e., ART based on Neural Networks, and HMM based on stochastic dynamics.

ART neural networks were developed by Carpenter and Grossberg [10] to understand the clustering function of the human visual system. They are based on a competitive learning scheme and are designed to deal with the stability/plasticity dilemma in clustering and general learning. ART neural networks successfully resolve this dilemma by matching the input pattern with the prototypes. If the matching is not adequate, a new prototype is created. In this way, previously learned memories are not eroded by new learning. In addition, the ART neural network implements a feedback mechanism during learning to enhance stability. Our experiments [23, 13] of using ART neural networks with combat simulation paths have been quite successful. We believe further improvement with the ART structure can lead to a fundamental breakthrough in large data clustering, which is needed in complex systems modelling. We have also proposed a heuristic that allows the magnitude of the input pattern to play a role in the clustering function. Furthermore,we are developing a generic numerical clustering tool, based on the ART neural network, that can be used for many important problems in intelligent data analysis.

In general, the description of a typical sample path generated by a discrete-event system requires a large amount of data since such sample paths are typically quite long. This implies that the dimension of each input pattern will also be large. However for high dimensional data most of the clustering algorithms (including ART) will involve huge computational effort; thus they are not practical for simulation modelling purposes. For this reason we develop a new clustering approach where we try to take advantage of the statistical structure behind a typical sample path. For high dimensional complex systems we try to use a Hidden Markov Model (HMM), which has been successfully used in speech recognition and other areas [69] to characterize each observed sample path. In our approach, we use an HMM to describe an arbitrary sample path and thus we cluster together all sample paths whose corresponding HMMs have a high similarity measure (to be defined in Section 5.2). The advantage of this approach is that the amount of data required to describe an HMM is generally much smaller than the amount of data required to explicitly describe an observed sample path and as a result the HMM approach is more efficient.

5.2 Hidden Markov Model as a Clustering Tool

A sample path generated by a discrete-event system consists of a sequence (e_k, t_k), $k = 1, 2, \ldots, K$, where e_k denotes the kth event and t_k its corresponding occurrence time. For typical systems, the number of observed events K is very large and thus attempting to cluster sample paths "directly", i.e., by making explicit use of the entire event sequence (e_k, t_k), requires the input vectors to be of a very large dimension which has an adverse effect on the computational requirements of most clustering tools (including ART). To address this problem, we observe a sample path that is generated by an arbitrary system and try to describe it by some Markov Chain, and thus we use the theory of Hidden Markov Models (HMMs) to identify its parameters. Once we identify the HMM parameters we define a similarity measure among each obtained HMM and cluster together all sample paths with the largest similarity. The advantage of this approach is that the amount of information required to describe an HMM is considerably less than the amount of information required to describe a sample path. Even though the identification of the HMM parameters requires some additional computational overhead, our experiments have shown that in overall, the

HMM approach is considerably faster than direct clustering approaches. Incidentally, we point out that this approach makes no a priori assumptions about the statistical distribution of the data to be analyzed.

5.2.1 Experimental Design

Next, we demonstrate the HMM clustering approach through an example. For the purposes of our example, we assume that we have three systems S_1 ,S_2 and S_3. When simulated, each system generates sample paths Q_{ij}, $i = 1, 2, 3, j = 1, 2, \ldots$, where Q_{ij} corresponds to the jth sample path generated by system S_i. When clustering sample paths, it would be reasonable to expect that sample paths generated from the same system are grouped in the same cluster.

In this example, we generate 9 sample paths, 3 from each system and develop a way to distinguish between sample paths obtained from different systems. To achieve this, we first associate an HMM $\lambda_i = (A_i, B_i, \pi_i)$ to each sample path $i = 1, \ldots, 9$, where we use the notation of Rabiner [69]. A_i denotes the state transition probability, B_i denotes the observation symbol probability at every state and π_i denotes the initial state distribution. We assume that it consists of N states and that for each state we can observe any of the M possible symbols.

To construct the three systems $S_i, i = 1, 2, 3$, we assume that they consist of a Markov Chain with N_i states ($N_1 = 20, N_2 = 10$, and $N_3 = 10$) and randomly generate a state transition probability matrix $P_i = [p^i_{kl}], k, l = 1, 2, \ldots, N_i$. Furthermore, we randomly generate the parameters $\mu^i_k, k = 1, 2, \ldots, N_i$, so that the exponentially distributed sojourn time for state k has mean $1/\mu^i_k$. However, not all real systems are memoryless, therefore, to make our example more interesting we introduce some “special” states, where state transitions out of such states are not made according to the state transition probability but rather through the set of rules we describe next.

For S_1, we assume that states $1, \ldots, 5$ are special states defined by the following rules:

- State 1: System stays at state 1 for either 1 or 3 consecutive, exponentially distributed sojourn time intervals, each with mean $1/\mu^1_1$. Then it jumps to state 10.
- State 2: System stays at state 2 a deterministic sojourn time interval of length $2/\mu^1_2$. Then it jumps to state n according to the state visited before arriving at state 2, denoted by S_{-1}:

$$n = \begin{cases} 15 & \text{if } S_{-1} \in \{1, \ldots, 10\} \\ 4 & \text{if } S_{-1} \in \{11, \ldots, 20\} \end{cases} \tag{5.1}$$

- State 3: System stays at state 3 for either 3 or 5 consecutive sojourn time intervals, each exponentially distributed with mean $1/\mu^1_3$. Then it transfers according to state transition probability matrix P_1.

- State 4: System stays at state 4 for either 4 or 1 exponentially distributed sojourn time intervals with mean $1/\mu_4^1$. Then it jumps to state n according to the state previously visited S_{-1}:

$$n = \begin{cases} 6 & \text{if } S_{-1} \in \{1,\dots,5\} \\ 11 & \text{if } S_{-1} \in \{6,\dots,10\} \\ 16 & \text{if } S_{-1} \in \{11,\dots,15\} \\ 1 & \text{if } S_{-1} \in \{16,\dots,20\} \end{cases} \tag{5.2}$$

- State 5: System stays at state 5 for a deterministic sojourn interval of length $1/\mu_5^1$,and then transfers according to the state transition probability matrix P_1.

Both S_2 and S_3 have only 2 special states defined by the following rules:

- State 1: System stays at state 1 or 2 sojourn time interval, each of which is exponentially distributed with mean $1/\mu_1^i, i = 2, 3$. Then it transfers to state n according to the previous state visited:

$$n = \begin{cases} 7 & \text{if } S_{-1} \in \{1,2\} \\ 9 & \text{if } S_{-1} \in \{3\} \\ 5 & \text{if } S_{-1} \in \{4,\dots,10\} \end{cases} \tag{5.3}$$

- State 2: System stays at state 2 or a deterministic amount of time equal to $1/\mu_2^i, i = 2, 3$, and then transfers to state 5.

Using S_1, S_2 and S_3, we generate 9 sample paths (3 from each system) and for each sample path we estimate the HMM model parameters $\lambda_j, j = 1, \dots, 9$, to maximize the probability that the jth observed sample path was obtained from λ_j. This is referred to as the training problem and is tackled by repeatedly solving what is described as "Problem 3" in Rabiner [69]. For the purposes of this experiment, we assume that each HHM consists of $N = 6$ states. In addition, we assumed that the actual state visited by each of the systems is not observable. Rather, the observation symbols at each state are the state holding times. These can generally take any positive values. To determine B_i, the symbol output probability, we quantize all possible values into $M = 64$ intervals.

Once we determine the HMM parameters for all sample paths, $\lambda_i, i = 1, \dots, 9$, we use the similarity measure defined in Rabiner [69] to determine which HMMs and consequently which sample paths are sufficiently similar so that they can be clustered together. The similarity measure is defined for any pair of HMMs λ_i and λ_j as:

$$\sigma(\lambda_i, \lambda_j) = \exp\{D(\lambda_i, \lambda_j)\} \tag{5.4}$$

where

$$D(\lambda_i, \lambda_j) = \frac{\log Pr(O^j|\lambda_i) + \log Pr(O^i|\lambda_j) - \log Pr(O^i|\lambda_i) - \log Pr(O^j|\lambda_j)}{2TK} \tag{5.5}$$

$\sigma(i,j)$	HMM1	HMM2	HMM3	HMM4	HMM5	HMM6	HMM7	HMM8	HMM9
HMM1	1	0.760	0.769	0.776	**0.950**	**0.950**	0.798	0.804	0.794
HMM2	0.760	1	**0.940**	**0.949**	0.772	0.776	0.837	0.839	0.835
HMM3	0.769	**0.940**	1	**0.947**	0.777	0.785	0.850	0.847	0.847
HMM4	0.776	**0.949**	**0.947**	1	0.787	0.793	0.847	0.844	0.844
HMM5	**0.950**	0.772	0.777	0.787	1	**0.951**	0.799	0.804	0.799
HMM6	**0.950**	0.776	0.785	0.793	**0.951**	1	0.815	0.820	0.809
HMM7	0.798	0.837	0.850	0.847	0.799	0.815	1	**0.945**	**0.943**
HMM8	0.804	0.839	0.847	0.844	0.804	0.820	**0.945**	1	**0.950**
HMM9	0.794	0.835	0.847	0.844	0.799	0.809	**0.943**	**0.950**	1

Table 5.1: Similarity measure between HMMs corresponding to each of the 9 sample paths

is what Rabiner [69] called the *distance measure*. $Pr(O^i|\lambda_j)$ is the probability of the observation sequence O^i, i.e., the sequence of state holding times that corresponds to the sample path Q_i, was generated by HMM λ_j. For computational convenience, we break any sample path into K segments of length T and thus compute

$$\log Pr(O^j|\lambda_i) = \sum_{k=1}^{K} \log Pr(O_k^i|\lambda_j) \tag{5.6}$$

where $Pr(O_k^i|\lambda_j)$ is the probability of the kth subsequence of sample path i was generated by HMM λ_i. Also, note that the similarity measure is symmetric, that is $\sigma(\lambda_i, \lambda_j) = \sigma(\lambda_j, \lambda_i)$; a desired property for a good similarity measure.

5.2.2 Experiments Results

In the similarity results shown in Table 5.1, the length of each of the 9 sample path is 10,000 events. In addition, the parameters $\mu_j^i, i = 1, 2, 3, j = 1, \ldots, N_i$ are generated such that $1/\mu_j^i$ are uniformly distributed between 4 and 50. Sample paths Q_1, Q_5, and Q_6 are generated by S_1. Q_2, Q_3, and Q_4 are generated by S_2 while Q_7, Q_8, and Q_9 are generated by S_3.

Finally, we cluster together all sample paths that correspond to HMMs with similarity measure greater than a threshold value V. Note that V corresponds to the required degree of similarity for two sample paths to be clustered together. For example, if $V = 0.9$, then the similarity measures exceeding V are:

Cluster 1: $\sigma(1,5), \sigma(1,6), \sigma(5,6)$
Cluster 2: $\sigma(2,3), \sigma(2,4), \sigma(3,4)$
Cluster 3: $\sigma(7,8), \sigma(7,9), \sigma(8,9)$

The resulting three clusters correspond to the generative models S_1,S_2 and S_3 respectively. Therefore, HMM has successfully classified all sample paths.

5.3 Application of Clustering in Computer Security

5.3.1 Motivation

Computer networks are complex systems which consist of multiple components. The interaction between human and computers, between servers and clients, between hardware and software, all contributes to the resulting complicated hierarchical structure of computer networks, either viewed physically or logically. Computer security defense, or intrusion detection in particular, aiming at identifying malicious activities performed by outside attackers or insider abusers by analyzing the behavior of computer networks via observable audit events, definitely needs the coordination of multiple detection sensors and components. This naturally leads to the application of the general framework of modelling and simulation of multi-resolution complex systems to computer security.

Intrusion activities in order to gain unauthorized access to system resources and file systems should leave traces at different levels of the system, but not necessarily significant at all levels. For example, the intrusion activity may appear normal at some levels of monitoring, and be distinct at another level. Hierarchical structural models are able to integrate both local and global information to make more accurate judgments.

We propose to apply clustering methods in system modelling for computer security. This is based on the observations of a hierarchical structure in both security audit data and user behavior, in which audit events from multiple sources with multiple resolutions are collected and analyzed. In this setting, clustering may help in characterizing the hierarchical structure for a better view and understanding on the system behavior, as it can in other multi-resolution complex systems.

Clustering is an unsupervised classification procedure, where the input data to the clustering algorithm are unlabelled, so it is not feasible to us to use it directly. Rather, we need to integrate domain knowledge as much as possible when clustering is introduced into the security system modelling.

5.3.2 Hierarchy in Computer Security Systems

Hierarchical structures are observed extensively in computer security systems, such as in [3] and [72]. We believe that both normal and abnormal system behavior should be described by integrating data from multiple sources at multiple levels.

Multi-resolution audit data

For security monitoring and intrusion detection, audit events of multiple resolution are collected from multiple sources as following:

- **system call traces**: collected from operating system kernel. They are the lowest level host-based audit data, and describe the execution of programs.

- **network packets**: collected from local network. They are the lowest level network-based audit data, describing the network traffic in and out the host.
- **command history data**: collected from shell or process table from operating system. They reside on the level above the system call traces, describing the observable behavior of users.
- **system events log**: a mixture of multi-resolution events, including messages or errors from applications and/or network protocols.

Multi-level system architecture

Based on the hierarchical structure of audit events, we describe the hierarchical structure of a multi-level host-based system model. For simplicity, we only consider the setting of a single host, i.e., we assume that all audit data are coming from a single monitored host. The five levels from the top to the bottom are host, user, operation, command, and system call. This structure is briefly illustrated in Fig 5.4, where operation level (not shown) is overlapping with user level.

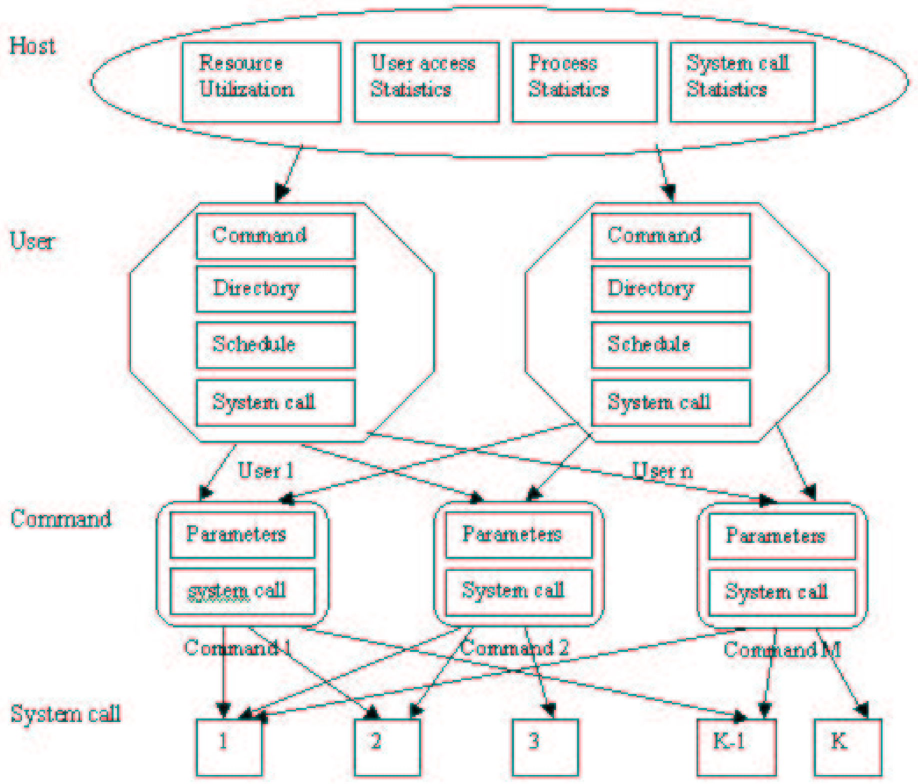


Figure 5.4: The hierarchical structure of computer security models

1. **Host level**: The top level in our architecture. In essence, the behavior of a host is characterized by three aspects of behavior: program behavior, user behavior and

network traffic behavior. In practice, we may observe the following aspects of system behavior:

- System Configuration: Change of system configuration may be an indication of anomaly.
- File Integrity: File integrity check provides valuable information on what really happened on the host
- Traffic Density: Dramatic change of network traffic density may suggest anomaly activities happening on the system.
- Resource Utilization: For each aspects of system resource such as CPU, memory and file access, we can record down the density of usage periodically. Using these records, we can construct a probability distribution of short-term and long-term behaviors for each of the resources.
- User access statistics: For each user who have access to the host, we can record the normal usage frequency and time schedule of users, and then construct a probability distribution over the user ID.
- Process statistics: For one given host, the statistics of process running over a period of time can be obtained. We can construct a probability distribution over the process name/group.
- System call statistics: For one given host, the statistics of system calls called over a period of time can be obtained. For each (or each group) of system calls, we can then construct the probability distribution over their name/functionality.

The above features are extracted from different levels of audit data. If we describe the behavior of a host by them and compute a numerical index, then a significant deviation of this index for a given period of time may result from anomaly behavior.

2. **User level**: Normally, intrusion is carried out directly by a user or by a process automatically run from a user's computer. The data at this level is user history data of commands and system calls collected on a single host. Sometimes it is difficult to distinguish two users who have similar work schedules and similar temporary tasks. So it may be useful to group the users by their main activities, such as the group for programming, the group for system administration, etc. Users can also be put into groups according to their proficiency, such as beginning learners, standard users and innovative experts. Actually, this level sometimes interleaves with the Operation Level, as we will see below.

3. **Operation level**: This level is also called activity level. It can be considered as a sub layer of the user level, or the layer parallel/overlapping with the user level. When working on a computer, each user may do one or more of the following tasks: Programming, System administration, Network utilities, Entertainment, Idle, etc. Normally, users' activities may interleave with each other, so it is not easy to recognize each distinct activity stream from a user's mixed command trace.

4. **Command level**: This level is equivalent to the program/functionality level, which is above the system call level and is the basic interface between the user and the host.

People suggests that profiling functionality or program execution is more effective than profiling individual users. The fewer number of functionality profiles are much more stable than users' profiles.

5. **System call level**: This level is also called Module Level, which is the lowest level of system behavior above the machine code. This level is the interface between the user application and the kernel of the operating systems. In Linux system, there are nearly 200 system calls. In our analysis, they each can be treated as a distinct state, or they can be grouped by their functionality. Statistics or structural models can be obtained at this level.

With this hierarchical structure in mind, we need clustering for Data Reduction and Event Correlation, to get representative models for characterizing normal profiles.

5.3.3 Characterizing the Hierarchical Structure

In this section, we discuss three different types of models used in computer security systems for the purpose of intrusion detection. We argue that the effectiveness of these models rely on the design of interfaces between high-resolution and low-resolution modules.

Modelling Program Behavior

One way to characterize the system behavior is to concentrate on the program behavior, because it is the abnormal program execution branches/exceptions that essentially leads to the unauthorized access or abuse of the system.

To describe program behavior, we need to look into lower level audit data, i.e., the execution traces of system calls. In this case, system call traces are generated by high-resolution sensors, while the representative models of programs are of lower resolution. This corresponds to the interface between "command" and "system call" level as shown in Fig 5.4. The task here is to summarize the data of system call traces into representative models of programs.

Various modelling schemes have been proposed to handle this problem, including simply pattern matching (either fixed-length or variable-length), Markov Chain Models, Decision Tree, Probabilistic Networks, Machine Learning models, Neural Networks, Hidden Markov Models, etc.

Modelling User Behavior

Another way to characterize system behavior is to look at the behavior of individual users based on the fact that it is the users who are interacting with the computer, performing normal/abnormal activities. This corresponds to the interface between "user" level and "command level", where the high-resolution view is command history data, and the low-resolution view is the profile of user behavior.

There is rich information within command history data, such as the time stamp of the command, command name, switches, and arguments. People usually only look at the sequence of command names and build models similar to the modelling of program behavior. However, command history data have their own special characteristics which call for distinct treatment rather than being treated the same as system call sequences.

Dealing with this set of high-dimensional data is not easy, especially with data fields of hybrid types. One way to deal with it is to treat each dimension of the information separately with different models chosen specific for each dimension. After decision results are generated from each of the dimensions, we integrate them together to form the final decision. Another way to deal with it is try to define some integrated index to incorporate all of the information into a single vector for training and testing.

People have proposed modelling user behavior through command history data by different models, such as pattern matching, machine learning, Markov models, high-order Markov models, uniqueness models, etc. There are some inherent difficulties within this data set that set limitations on the performance of those models in intrusion detection, such as the high randomness within command history data, concept-drift problem addressing the shift of behavior over time, etc. A hybrid model taking advantage of both statistical-based components and signature-based components may be the possible solution to this problem for better performance.

Above "user level", we may add another level, "user group level". We build this level by doing clustering on user behavior models, and examine the commonty among users. User groups can thus be constructed with users of similar background, task domain, proficiency and working schedules. Such groups are indeed observed in user command history data. Here clustering is used for getting the representative models for user group.

Actually, we could insert another level, "activity level" above "command level", where clustering may be used for extracting the models for different type of user activities. Models can be built individually for different activities, such as editing documents, checking email, system configuration and maintenance, entertainment activity, etc., to investigate the underlying dynamics below the command sequences. These streams of activities may interleave with each other if viewed from command history data. It remains a question whether we can find good representative models for those activities.

Modelling Network Traffic Behavior

The third angle of view on the system behavior is through the behavior of network traffic. There are many network protocols and services running in current computer network setting, such as http, ftp, telnet, SMTP, SNMP, ICMP, DNS, etc. Without the communications built by network protocols and the vulnerabilities in the design and implementation of them, most intrusions could never be carried out. Analyzing the profiles of network traffic data is especially important in dealing with Distributed Denial of Service (DDoS) attack which has become the most serious threat to Web servers on Internet.

In this context, the high-resolution data are the network packets, and the low-resolution models are describing the behavior of different network protocols or different network operation scenarios. This interface is not shown in Fig 5.4, but it is critical to the successful coordination among different computers on the network to defense attacks.

People have proposed different techniques to modelling network traffic data to describe the normal profiles of network, to support IP Traceback, to detect stepping-stones for DDoS attack, to characterizing Worm propagation, etc.

5.3.4 Related Work

Clustering has been used in user modelling for anomaly detection. In [52], T. Lane proposes user modelling based on pattern matching on short segments of command history data. In this model, frequently used short segments of commands extracted from normal data constitute the normal profiles. For efficiency in both storage and matching, he uses clustering for data reduction on the patterns stored in normal database. In [58], J. Marin et al. proposed a hybrid model to profile user behavior by the relative frequency of command usage stored in vectors. They use k-means clustering for generating the initial reference vector.

Clustering has been shown to help in both data reduction and model representation. Clustering may also be used for event correlation when events coming from multiple sensors are gathered together for a comprehensive view on what is going on in the system. In our study shown next, we propose to use Hidden Markov model as a clustering tool at the interface between high-resolution audit data and low-resolution audit data.

5.4 Experiments on HMM for Anomaly Detection

5.4.1 Applications of HMM in Computer Security

We have shown that clustering may help in the modelling of system behavior for computer security, and HMM is a powerful candidate to do the clustering job. So here we will examine the possible applications of Hidden Markov Models in computer security. Due to its strong descriptive power, HMM can either be used for modelling user behavior, or program behavior. In [81], Hidden Markov Models are used for modelling program behavior through the execution traces of system call sequences and the performance is compared with some other simple models. The results show that Hidden Markov Models are powerful to describe the behavior of normal program in the context of anomaly detection, at the cost of computational burden in training. In [51], T. Lane chooses Hidden Markov Models for profiling normal user behavior represented by Unix shell command history sequences, where again HMMs are shown to be able to characterize the complicated structure within user command sequences.

On the other hand, HMM can be used in multiple levels of modelling, depending on the resolution of view. For example, it can be used for building a model for sample paths which are mixtures of multiple activities, or it can be built based on a specific activity that are previously extracted from the sample path.

We have done some experiments where we apply Hidden Markov Models to model the normal behavior of network traffic log and use this model as the reference to detect any anomalous behavior in the network log data. Next we will show some considerations during model construction, and give brief discussions on the experiments.

5.4.2 Model Construction

A set of experiments were performed on SIAC company security log data using Hidden Markov Models. The purpose of the experiments was to use Hidden Markov Models to characterize the normal behavior of the system traffic represented by audit event "sample path", so that the traffic caused by intrusive activities showing a large diverge from the normal model will be captured by the clustering procedure.

In SIAC log data, each line is a log event. One connection (such as an http connection) can generate one or more log events. Most of the connections (more than 99.8%) are http connections, with small percentage of other events, such as ftp, smap, sendmail, etc.

We have made some necessarily modifications to the HMM to make it more precisely in describing user behavior through network traffic log data.

1. **Determining number of states**: The state in our model represents an abstract relatively stable status of a computer user, corresponding to the users' main activity within a given period of time. Viewed from audit data, each state corresponds to different subgroup of events. For example, when user's current state is "programming", the *dominant* events will be editing/compiling; when the state changes to "surfing web", *dominant* events will be related to HTTP.

2. **Two Critical states**: We choose "login" as an initial state of the HMM model. The initial state is the entry state that records some important information of the user, such as user name, login time, source IP address, login failure times, etc. Among all the intrusion actions, a large percentage of them are to gain unauthorized root access into a local computer system, either from a local user account or directly from outside. So we add a state called "user-to-root" state to provide information about how root privilege is obtained. In the overall HMM model matching algorithm, we put heavier weight on the matching score of this state. If a user usually logs in as a supervisor, then the "user-to-root" state is the same as the initial state.

3. **Data Segmentation**: To partition the audit events traces into discrete events, we take into account the fact that different state has different type of "dominant" commands. So we use "window" concept in our model. For example, if in the last 20 events the type of dominant events has changed from A to B, then we suspect that

the state has transited from state A to B. Here we apply two types of windows: time window and event counter window, which deal with time interval and event counters respectively.

4. **Feature selection**: The simplest way is to only record the distribution of different events in a state. For intrusion detection, this is surely not enough. In order to make more accurate detection, we suggest to use some rule-based detection techniques to add more features, Such as command duration time, state duration time (exponential distributed), overall command number, number of "hot actions" (e.g., access to system directories, creation and execution of programs, etc), number of access to "access control" files (e.g., `/etc/passwd`, `.rhosts`), etc.

5.4.3 Model Training

From above discussion, we have known that the number of states is pre-determined. Since we have partitioned the audit data sequences into separate parts, each part with its own "dominant" events, so it is easy to know which state the user transit to. It means that the states here are almost sure observable. So the state transition probability and state duration time can be easily calculated from training data.

For each state record, the distribution of commands can be easily obtained by calculated frequency. For other parameters, such as command duration time, hot actions number, accessing critical file numbers, we can treat them as Gaussian distribution. This approximation is reasonable and simple to implement. It should be noted that since most parameters in HMM have physical meaning, the system manager can set initial values to these parameters in advance. So the training task will be light-burdened and more efficient.

5.4.4 Calculating Similarity Measure

Since our HMM model has modified a lot from original model, the matching criterion is not as simple as calculating the probability of the observation sequence by the given model. Instead, we compute a "suspicious score" for the matching process.

The critical state, i.e., the initial state and "user-to-root" state has higher weights while other state has lower weights. Each state has its own suspicious rate score, computed by integrating the score of difference between each parameter and its observation value. Then we compare the suspicious score of the user with a threshold level. The threshold level is set to trade off false alarm and missed detections. Each parameter in the record of state is treated as a Gaussian distributed random variable.

Let J be the "suspicious score" of an HMM model. The model has K states. During preprocessing, the user's audit event sequences are divided into N parts, and so he has $N-1$ state transition. We calculate the "suspicious score" by:

$$J = W_1 S_1 + \sum_{k=2}^{N} \frac{1}{P_{i,j} W_j S_j^k} \tag{5.7}$$

where $P_{i,j}$ is the probability of state transition from state i to state j, W_j is the weight factor of state j (the "user-to-root" state and initial state have larger W_j), S_j^k is the suspicious score of state j compared with the kth observed state. For example, if a user's observed action sequence is $S_1S_2S_4S_3S_2$, then the suspicious score is:

$$J = W_1S_1 + \frac{1}{P_{1,2}}W_2S_2^2 + \frac{1}{P_{2,4}}W_4S_4^3 + \frac{1}{P_{4,3}}W_3S_3^4 + \frac{1}{P_{3,2}}W_2S_2^5 \tag{5.8}$$

Suppose there are M parameters in state i, each parameter $x_j, j = 1, \ldots, M$ has expectation ε_j and variance σ_j, the observed value of parameter is o_j, then S_i^k can be calculated as:

$$S_i^k = \sum_{j=1}^{M} \omega_j \frac{(o_j - \varepsilon_j)^2}{\sigma_j} \tag{5.9}$$

where ω_j is the weight factor of parameter x_j. The weight factors W_j and ω_j can be set by the system manager and be modified from training.

5.4.5 Discussion

SIAC log data contains two parts of data: normal and abnormal. In the abnormal part, there are some kinds of intrusion attempts within. Our mission is to locate these intrusion attempts in the abnormal part, with normal data as training data. This set of experiments are not successful. Here we will discuss why HMM failed in modelling with SIAC data.

1. **Insufficient information**: In preprocessing, too much information has been ignored. SIAC data contains mainly two parts of data, http connections and non-http connections. In normal data, the non-http connections only constitute less than 0.2%, so this part of data are not significant enough in preprocessing for statistical approximation. But the most likely intrusion here is in non-http connections, so only dealing with http connections is not enough.
   On the other hand, even for http connections, too much information has been ignored due to the potential burden due to the complexity of information format.
   Furthermore, if this set of log data itself does not contain enough information to discriminate between normal and abnormal behavior, there will be no way to find out the intrusions.

2. **Difficulty in quantization**: The preprocessed data of SIAC http connection are vectors of sequential data, but different terms in the vector have totally different numerical domain. The last three attributes are of in the range of $1, 2, 3$, while the first four attributes vary from 0 to 10^6. The vector elements must be quantized into the same domain. But here it is difficult to do it without a suitable Vector Quantization method.

3. **Common problems with statistics based anomaly detection**: For statistical based anomaly detection to be successful, we have some basic assumptions: First, it requires statistically enough normal behavior data for training to be able to cover the variance of normal behavior, which sometimes is very difficult to achieve; Second, in order to find out whether there are intrusions in a segment of test data, the test data must also be rich enough for statistical based analysis. Otherwise, there will not be enough statistical information in the test data segment to justify the results.
   On the other hand, if we make the test data segment long enough for statistical analysis, we may be able to only determine whether there are abnormal behaviors in this data segment, but not able to tell where they are and what kind of intrusion it is. This part of workload is left for rule-based intrusion detection system or human experts.

5.5 Summary

In this chapter, we have discussed the applications of clustering methods in hierarchical simulation of complex systems and in system modelling for computer security. We propose to use clustering techniques between high- and low-resolution modules for the analysis and simulation of multi-resolution to preserve the statistics fidelity. We demonstrated that Hidden Markov Model can be an effective clustering tool for this task. Then we shift to the possible applications of clustering techniques in the domain of computer security based on the observation of hierarchical structure computer security systems. Three aspects of behavioral models are discussed with the possible applications of clustering as the important components between different levels of audit events. We then attempted to use Hidden Markov Models for the purpose of system modelling for anomaly detection, discussed several issues related to this problem and possible solutions.

Chapter 6

OPTIMIZATION EXAMPLES

In this chapter we will use the results from the concurrent simulation chapter for control and optimization. We will first use the IPA estimates for performing management functions in a single node in a communication network. Subsequently, we use concurrent estimation [21] together with the "surrogate" methodology [35, 32] to perform multi-commodity resource allocation in the context of mission planning in a Joint Air Operation (JAO) environment.

6.1 Optimal Buffer Control Using SFM-Based IPA Estimators

We consider here an optimization problem for single-node SFMs involving loss volume and workload levels; both are network-related performance metrics associated with buffer control or call-admission control. In a typical buffer control problem, for instance, the optimization problem involves the determination of a threshold (measured in packets or bytes) that minimizes a weighted sum of loss volume and buffer content. One possible problem formulation is to determine a threshold C that minimizes a cost function of the form

$$J_T(C) \; = \; \bar{Q}_T(C) \; + R \cdot \bar{L}_T(C)$$

trading off the expected loss rate with a rejection penalty R for the expected queue length. If a SFM is used instead, then the cost function of interest becomes

$$J(\theta) = \frac{1}{T}E[Q_T(\theta)] + \frac{R}{T}E[L_T(\theta)].$$

In the case of the simple buffer control problem, we are interested in estimating $dJ_T/d\theta$ based on directly observed (simulated) data. We can then seek to obtain θ^* such that it minimizes $J_T(\theta)$ through an iterative scheme of the form

$$\theta_{n+1} = \theta_n - \nu_n H_n(\theta_n, \omega_n^{SFM}), \qquad n = 0, 1, \ldots \tag{6.1}$$

where $\{\nu_n\}$ is a step size sequence and $H_n(\theta_n, \omega_n^{SFM})$ is an estimate of $dJ_T/d\theta$ evaluated at $\theta = \theta_n$ and based on information obtained from a sample path of the SFM denoted by ω_n^{SFM}. However, as we saw in Chapter 3, the simple form of $H_n(\theta_n, \omega_n^{SFM})$ also enables us to apply the same scheme to the original discrete event system:

$$C_{n+1} = C_n - \nu_n H_n(C_n, \omega_n^{DES}), \qquad n = 0, 1, \ldots \tag{6.2}$$

where C_n is the threshold used for the nth iteration and ω_n^{DES} is a sample path of the discrete event system.

The gradient estimator $H_n(\theta, \omega_n^{SFM})$ is the IPA estimator of $dJ/d\theta$ based on (3.9) and (3.10):

$$H_n(\theta, \omega_n^{SFM}) = \frac{1}{T} \sum_{k \in \Phi(\theta)} [\eta_k(\theta) - u_{k,1}(\theta)] - \frac{R}{T} B(\theta) \tag{6.3}$$

evaluated over a simulated sample path ω_n^{SFM} of length T, following which a control update is performed through (6.1) based on the value of $H_n(\theta, \omega_n^{SFM})$. The interesting observation here is that the same estimator may be used in (6.2) as follows: If a packet arrives and is rejected, the time this occurs is recorded as τ in Algorithm 1. At the end of the current busy period, the counter $\mathcal{C}$ and timer $\mathcal{T}$ are updated. Thus, the exact same expression as in the right-hand side of (6.3) can be used to update the threshold C_{n+1} in (6.2).

Figure 6.1 depicts examples of the application of this scheme to a single-node SFM under six different parameter settings (scenarios), summarized in Table 6.1. 'DES' denotes curves obtained by estimating $J_T(C)$ over different (discrete) values of C, 'SFM' denotes curves obtained by estimating $J(\theta)$ over different values of θ, and 'Opt.Algo.' represents the optimization process (6.2), where we maintain real-valued thresholds throughout. The first three scenarios correspond to a high traffic intensity ρ compared to the remaining three. For each example, C^* is the optimal threshold obtained through exhaustive simulation. In all simulations, an ON-OFF traffic source is used with the number of arrivals in each ON period geometrically distributed with parameter p and arrival rate α; the OFF period is exponentially distributed with parameter μ; and the service rate is fixed at β. Thus, the traffic intensity of the system ρ is $\alpha(\frac{1}{\alpha p})/\beta(\frac{1}{\alpha p} + \frac{1}{\mu})$, where $\frac{1}{\alpha p}$ is the average length of an ON period and $\frac{1}{\mu}$ is the average length of an OFF period. The rejection cost is $R = 50$. For simplicity, ν_n in (6.2) is taken to be a constant $\nu_n = 5$. Finally, in all cases $T = 100,000$. As seen in Fig. 6.1, the threshold value obtained through (6.2) using the SFM-based gradient estimator in (6.3) either recovers C^* or is close to it with a cost value extremely close to $J_T(C^*)$; since in some cases the cost function is nearly constant in the neighborhood of the optimum, it is difficult to determine the actual optimal threshold, but it is also practically unimportant since the cost is essentially the same. We have also implemented (6.2) with $H_n(C_n, \omega_n^{DES})$ estimated over shorter interval lengths $T = 10,000$ and $T = 5,000$, with virtually identical results. Looking at Fig. 6.1, it is worth observing that determining θ^* as an approximation to C^* through off-line analysis of the SFM would also yield good approximations, further supporting the premise of this chapter that SFMs provide an attractive modeling framework for control and optimization (not just performance analysis) of complex networks.

Scenario	ρ	α	p	μ	β	C^*
1	0.99	1	0.1	0.1	0.505	7
2	0.99	1	0.05	0.05	0.505	7
3	0.99	2	0.05	0.1	1.01	15
4	0.71	1	0.1	0.1	0.7	13
5	0.71	1	0.05	0.05	0.7	11
6	0.71	2	0.05	0.1	1.4	22

Table 6.1: Parameter settings for six examples

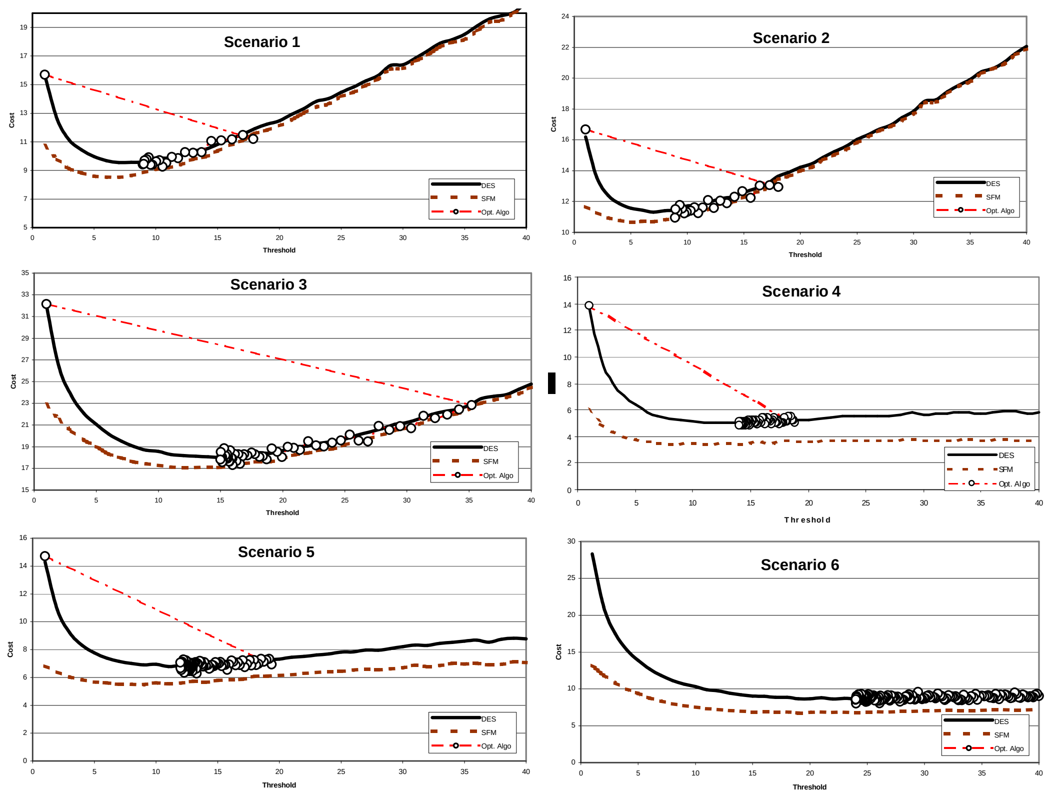


Figure 6.1: Optimal threshold determination in an actual system using SFM-based gradient estimators - Scenarios 1-6

6.2 Multi-commodity Resource Allocation

In this section we investigate the problem of mission planning in the context of the Joint Air Operation (JAO) environment. For this problem we assume that there are N_i type i aircraft that can be used in any given mission. For simplicity we assume that there are only two types of aircraft, $i = 1$ (strike aircraft) or $i = 2$ (wild weasel). The objective is to dynamically allocate these aircraft to various missions against a set of predefined targets. We assume that each target is destroyed probabilistically and the probability is monotonically increasing with the number of strike aircraft allocated to it. Furthermore, when destroyed, each target carries a value that indicates its significance. In addition, each target j is generally defended by a set of n_j SAMs that constitute risk for the mission aircraft. Every destroyed aircraft incurs a cost c_i while destroyed SAM sites constitute no quantifiable benefit. Finally, again for simplicity, we assume that the path that the mission will follow is the straight line between the aircraft base and the target. The problem is to dynamically schedule various missions until either all targets are destroyed or there are not enough aircraft to take up a new mission. The objective is to maximize the expected value obtained from all destroyed targets minus the cost of lost aircraft. This problem formulation leads to a non-convex combinatorialy hard problem which we solve using a "surrogate" methodology which is briefly described next (For more details the reader is referred to Appendix D or [35, 32]).

6.2.1 Basic Approach for the "Surrogate" Method

We define an optimization problem the general form

$$\min_{r \in A_d} J_d(r) = E[L_d(r, \omega)] \tag{6.4}$$

where $r \in \mathbb{Z}_+^N$ is a decision vector or "state" and A_d represents a constraint set. In a stochastic setting, let $L_d(r, \omega)$ be the cost incurred over a specific sample path ω when the state is r and $J_d(r) = E[L_d(r, \omega)]$ be the expected cost of the system operating under r. The sample space is $\Omega = [0, 1]^\infty$, that is, $\omega \in \Omega$ is a sequence of random numbers from $[0, 1]$ used to generate a sample path of the system. The cost functions are defined as $L_d : A_d \times \Omega \to \mathbb{R}$ and $J_d : A_d \to \mathbb{R}$, and the expectation is defined with respect to a probability space $(\Omega, \Im, P)$ where $\Im$ is an appropriately defined σ-field on Ω and P is a conveniently chosen probability measure. In the sequel, 'ω' is dropped from $L_d(r, \omega)$ and, unless otherwise noted, all costs will be over the same sample path.

Let the expected cost function $J_d(r)$ is generally nonlinear in r, a vector of integer-valued decision variables, therefore (6.4) is a nonlinear integer programming problem. One common method for solving this problem is to relax the integer constraints on all r_i so that they can be regarded as continuous (real-valued) variables and then to apply standard optimization techniques such as gradient-based algorithms. Let the "relaxed" set A_c contain the original constraint set A_d and define $\bar{L}_c : \mathbb{R}_+^N \times \Omega \to \mathbb{R}$ to be the cost function over a specific sample path. The resulting "surrogate" problem then becomes: Find ρ^* that minimizes the "surrogate" expected cost function $J_c : \mathbb{R}_+^N \to \mathbb{R}$ over the continuous set A_c, i.e.,

$$J_c(\rho^*) = \min_{\rho \in A_c} J_c(\rho) = E[\bar{L}_c(\rho)] \tag{6.5}$$

where $\rho \in \mathbb{R}_+^N$, is a real-valued state, and the expectation is defined on the same probability space $(\Omega, \Im, P)$ as described earlier. Assuming an optimal solution ρ^* can be determined, this state must then be mapped back into a discrete vector by some means (usually, some form of truncation). Even if the final outcome of this process can recover the actual r^* in (6.4), this approach is strictly limited to *off-line* analysis: When an iterative scheme is used to solve the problem in (6.5) (as is usually the case except for very simple problems of limited interest), a sequence of points $\{\rho_n\}$ is generated; these points are generally continuous states in A_c, hence they may be infeasible in the original discrete optimization problem. Moreover, if one has to estimate $E[\bar{L}_c(\rho)]$ or $\frac{\partial E[\bar{L}_c(\rho)]}{\partial \rho}$ through simulation, then a simulation model of the surrogate problem must be created, which is also not generally feasible. If, on the other hand, the only cost information available is through direct observation of sample paths of an actual system, then there is no obvious way to estimate $E[\bar{L}_c(\rho)]$ or $\frac{\partial E[\bar{L}_c(\rho)]}{\partial \rho}$, since this applies to the real-valued state ρ, not to the integer-valued actual state r.

Here we adopt a different approach intended to operate *on line*. In particular, we still invoke a relaxation such as the one above, i.e., we formulate a surrogate continuous optimization problem with some state space $A_c \subset \mathbb{R}_+^N$ and $A_d \subset A_c$. However, at every step n of the iteration scheme involved in solving the problem, both the continuous and the discrete states are simultaneously updated through a mapping of the form $r_n = f_n(\rho_n)$. This has two advantages: First, the cost of the original system is continuously adjusted (in contrast to an adjustment that would only be possible at the end of the surrogate minimization process); and second, it allows us to make use of information typically needed to obtain cost sensitivities from the actual operating system at every step of the process.

Initially, we set the "surrogate system" state to be that of the actual system state, i.e.,

$$\rho_0 = r_0 \tag{6.6}$$

Subsequently, at the nth step of the process, let $H_n(\rho_n, r_n, \omega_n)$ denote an estimate of the sensitivity of the cost $J_c(\rho_n)$ with respect to ρ_n obtained over a sample path ω_n of the actual system operating under allocation r_n. Two sequential operations are then performed at the nth step:

1. The continuous state ρ_n is updated through

$$\rho_{n+1} = \pi_{n+1}[\rho_n - \eta_n H_n(\rho_n, r_n, \omega_n)] \tag{6.7}$$

where $\pi_{n+1} : \mathbb{R}^N \to A_c$ is a projection function so that $\rho_{n+1} \in A_c$ and η_n is a "step size" parameter.

2. The newly determined state of the surrogate system, ρ_{n+1}, is transformed into an actual feasible discrete state of the original system through

$$r_{n+1} = f_{n+1}(\rho_{n+1}) \tag{6.8}$$

where $f_{n+1} : A_c \rightarrow A_d$ is a mapping of feasible continuous states to feasible discrete states which must be appropriately selected as will be discussed later.

One can recognize in (6.7) the form of a stochastic approximation algorithm (e.g., [50]) that generates a sequence $\{\rho_n\}$ aimed at solving (6.5). However, there is an additional operation (6.8) for generating a sequence $\{r_n\}$ which we would like to see converge to r^* in (6.4). It is important to note that $\{r_n\}$ corresponds to feasible realizable states based on which one can evaluate estimates $H_n(\rho_n, r_n, \omega_n)$ from observable data, i.e., a sample path of the actual system under r_n (not the surrogate state ρ_n). We can therefore see that this scheme is intended to combine the advantages of a stochastic approximation type of algorithm with the ability to obtain sensitivity estimates with respect to discrete decision variables. In particular, the sensitivity estimation methods studied in Chapter 3 are ideally suited to meet this objective.

6.2.2 Optimization Algorithm

In this section we simply summarize the algorithm used to solve the basic problem in (6.4) (For more details refer to [35, 32]).

Algorithm 2

Step 0. *Initialize* $\rho_0 = r_0$ *and perturb* ρ_0 *to have all components non-integer.*

For any iteration $n = 0, 1, \ldots$

Step 1. *Determine* $\mathcal{S}(\rho_n)$ *[using the construction of Theorem 3.1 [32]; recall that this set is generally not unique].*

Step 2. *Select* $f_n \in \mathcal{F}_{\rho_n}$ *such that* $r_n = \arg\min_{r \in \mathcal{N}(\rho_n)} \|r - \rho_n\| = f_n(\rho_n) \in \mathcal{N}(\rho_n)$.

Step 3. *Operate at* r_n *to collect* $L_d(r^i)$ *for all* $r^i \in \mathcal{S}(\rho_n)$ *[using Concurrent Estimation or some form of Perturbation Analysis; or, if feasible, through off-line simulation].*

Step 4. *Evaluate* $\nabla L_c(\rho_n)$.

Step 5. *Update the continuous state:* $\rho_{n+1} = \pi_{n+1}[\rho_n - \eta_n \nabla L_c(\rho_n)]$.

Step 6. *If some stopping condition is not satisfied, repeat Steps 1-6 for* $n+1$. *Else, set* $\rho^* = \rho_{n+1}$.

Step 7. *Obtain the optimal (or the near optimal) state as one of the neighboring feasible states in the set* $\mathcal{N}(\rho^*)$.

Note that for separable cost functions, Steps 1-6 can be replaced by

Step 1. Select f_n such that $r_n = \arg\min_{r \in \mathcal{N}(\rho_n)} \|r - \rho_n\| = f_n(\rho_n) \in \mathcal{N}(\rho_n)$.

Step 2. Operate at r_n to evaluate $\nabla L_c(\rho_n)$ using Perturbation Analysis.

Step 3. Update the continuous state: $\rho_{n+1} = \pi_{n+1}[\rho_n - \eta_n \nabla L_c(\rho_n)]$.

Step 4. If some stopping condition is not satisfied, repeat Steps 1-4 for $n+1$. Else, set $\rho^* = \rho_{n+1}$.

Note that ideally we would like to have $\nabla J_c(\rho_n)$ be the cost sensitivity driving the algorithm. Since this information is not always available in a stochastic environment and since $J_c(\rho_n) = E[\bar{L}_c(\rho_n, \omega)]$, the stochastic approximation algorithm uses $\nabla \bar{L}_c(\rho_n, \omega)$ as an estimate and under some standard assumptions on the estimation error ε_n where

$$\nabla J_c(\rho_n) = \nabla \bar{L}_c(\rho_n, \omega) + \varepsilon_n$$

the convergence is guaranteed. In order to get $\nabla \bar{L}_c(\rho_n, \omega)$, however, one needs to consider all possible selection sets. In this algorithm we utilize only one of those selection sets and approximate $\nabla \bar{L}_c(\rho_n, \omega)$ with $\nabla L_c(\rho_n, S(\rho_n), \omega)$. This approximation introduces yet another error term $\bar{\varepsilon}_n$ where

$$\nabla \bar{L}_c(\rho_n, \omega) = \nabla L_c(\rho_n, S(\rho_n), \omega) + \bar{\varepsilon}_n$$

Note that this error term $\bar{\varepsilon}_n$ exists regardless of stochasticity, unless the cost function $L_d(.)$ is separable (all selection sets will yield the same sensitivity for separable cost functions). We can combine error terms to define $\epsilon_n = \bar{\varepsilon}_n + \varepsilon_n$ and write

$$\nabla J_c(\rho_n) = \nabla L_c(\rho_n, S(\rho_n), \omega) + \epsilon_n$$

If the augmented error term ϵ_n satisfies the standard assumptions, then convergence of the algorithm to the optimal follows.

6.2.3 Multicommodity Resource Allocation Problems

An interesting class of discrete optimization problems arises when Q different types of resources must be allocated to N users. The corresponding optimization problem we would like to solve is

$$\min_{r \in A_d} J(r)$$

where $r = [r_{1,1}, \ldots, r_{1,Q}, \cdots, r_{N,1}, \ldots, r_{N,Q}]$ is the allocation vector and $r_{i,q}$ is the number of resources of type q allocated to user i. A typical feasible set A_d is defined by the capacity constraints

$$\sum_{i=1}^{N} r_{i,q} \leq K_q, \quad q = 1, \ldots, Q$$

and possibly additional constraints such as $\beta_i \leq r_{i,q} \leq \gamma_i$ for $i = 1, \ldots, N$. Aside from the fact that such problems are of higher dimensionality because of the Q different resource types that must be allocated to each user, it is also common that they exhibit multiple local minima. Examples of such problems are encountered in mission planning that involve N

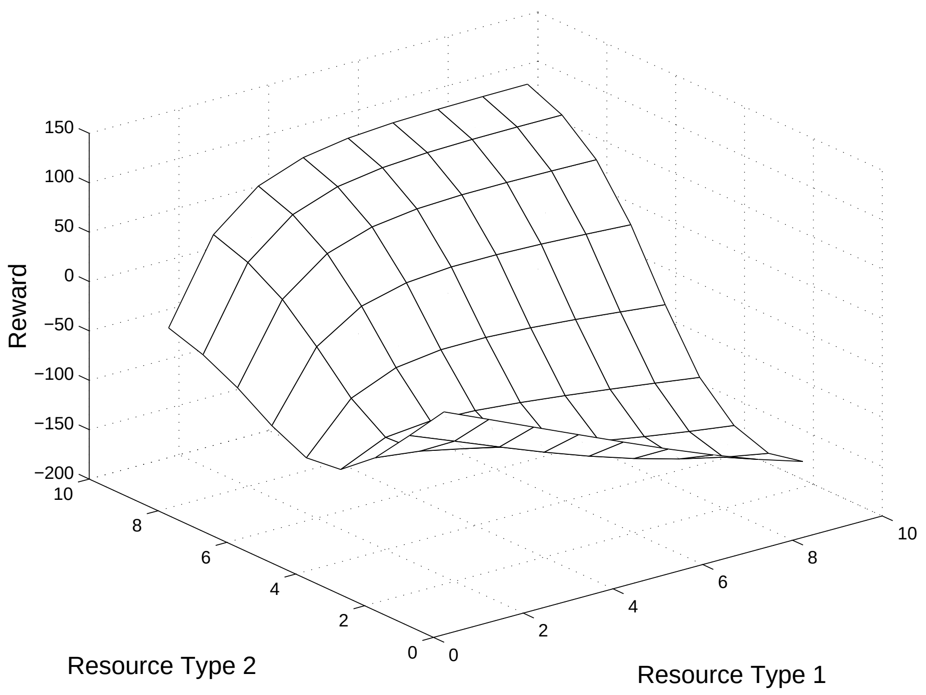


Figure 6.2: A typical reward function $J_i(r_{i,1}, r_{i,2})$.

missions to be simultaneously performed, each mission i requiring a "package" of resources $(r_{i,1}, \ldots, r_{i,Q})$ in order to be carried out. The natural trade-off involved is between carrying out fewer tasks each with a high probability of success (because each task is provided adequate resources) and carrying out more tasks each with lower probability of success.

The "surrogate problem" method provides an attractive means of dealing with these problems with local minima because of its convergence speed. Our approach for solving these problems is to randomize over the initial states r_0 (equivalently, ρ_0) and seek a (possibly local) minimum corresponding to this initial point. The process is repeated for different, randomly selected, initial states so as to seek better solutions. For deterministic problems, the best allocation seen so far is reported as the optimal. For stochastic problems, we adopt the stochastic comparison approach in [36]. The algorithm is run from a randomly selected initial point and the cost of the corresponding final point is compared with the cost of the "best point seen so far". The stochastic comparison test in [36] is applied to determine the "best point seen so far" for the next run. Therefore, the surrogate problem method can be seen as a complementary component for random search algorithms that exploits the problem structure to yield better generating probabilities (as discussed in [36]), which will eliminate (or decrease) the visits to poor allocations enabling them to be applied on-line.

In what follows we consider a problem with $N = 16$, $Q = 2$, and $K_1 = 20$, $K_2 = 8$. We then seek a 32−dimensional vector $r = [r_{1,1}, r_{1,2}, \cdots, r_{16,1}, \ldots, r_{16,2}]$ to maximize a reward function of the form

$$J(r) = \sum_{i=1}^{16} J_i(r) \tag{6.9}$$

subject to

$$\sum_{i=1}^{N} r_{i,1} \leq 20, \qquad \sum_{i=1}^{N} r_{i,2} \leq 8$$

The reward functions $J_i(r)$ we will use in this problem are defined as

$$J_i = V_i P_i^0(r) - C_1 r_{i,1} P_i^1(r) - C_2 r_{i,2} P_i^2(r) \tag{6.10}$$

In (6.10), V_i represents the "value" of successfully completing the ith task and $P_i^0(r)$ is the probability of successful completion of the ith task under an allocation r. In addition, C_q is the cost of a resource of type q, where $q = 1, 2$, and $P_i^q(r)$ is the probability that a resource of type q is completely consumed or lost during the execution of the ith task under an allocation r. A representative example of a reward function for a single task with $V_i = 150$ is shown in Fig. 6.2. The cost values of resource types are $C_1 = 20$ and $C_2 = 40$, and the values for tasks we will use in this problem range between 50 and 150.

The surrogate method is executed from random initial points and the results for some runs are shown in Fig. 6.3. Note that due to local maxima, some runs yield suboptimal results. However, in all cases convergence is attained extremely fast, enabling us to repeat the optimization process multiple times with different initial points in search of the global maximum. Although it is infeasible to identify the actual global maximum, we have compared our approach to a few heuristic techniques and pure random search methods and found the "surrogate problem" method to outperform them. To demonstrate the effectiveness of the approach we have developed an "applet" for the scenario described above which can be accessed at `http://vita.bu.edu/cgc/alpha/index.htm`.

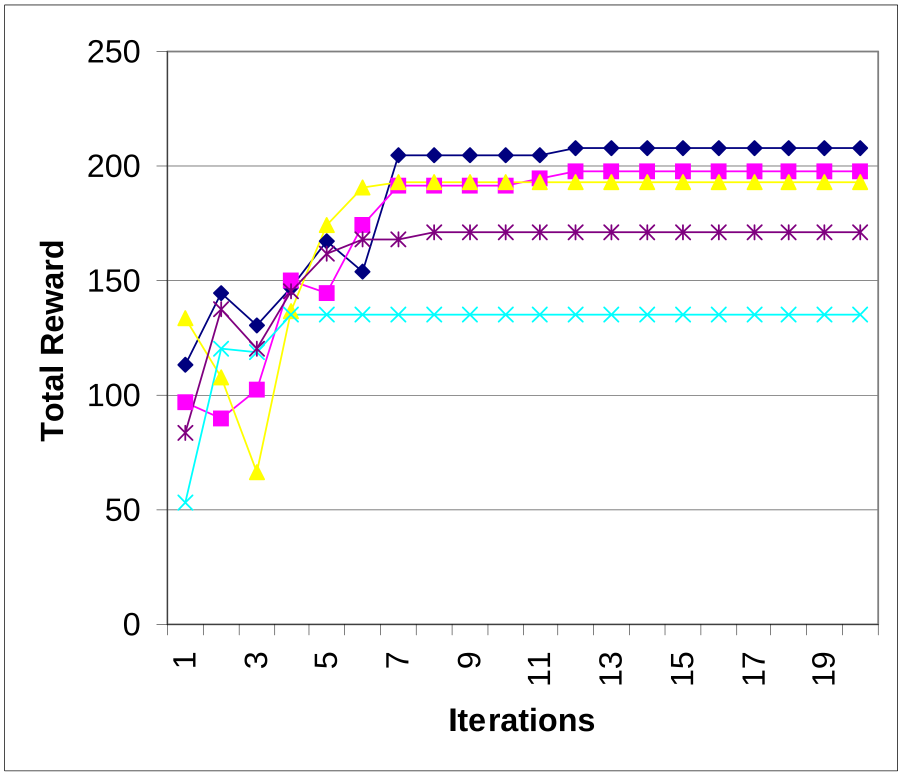


Figure 6.3: Algorithm convergence under different initial points.

Chapter 7

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

In this section, we summarize the main findings, lessons learned, and recommendations for future directions that have resulted from this project.

Model abstraction through fluid simulation: Fluid models (FM) are considered as efficient *abstract* modeling paradigms that can either approximate the dynamics of complex discrete-event systems or constitute primary models in their own right. In FM, the state of the system is described by discrete as well as continuous type variables. Furthermore, the system dynamics are both, time-driven and event-driven, therefore FM fall in the class of *hybrid* simulation models, which can be used to model a fairly broad class of systems including battle engagements, communication networks, manufacturing systems and many more.

For performance evaluation, when a FM is used to approximate the dynamics of a discrete-event system the important tradeoff is efficiency vs. accuracy. This tradeoff relates to the resolution (i.e., number of events to be aggregated together) of the fluid model used which is thoroughly investigated. On the other hand, FM can be used for the *control and optimization* of various systems. In this case, a FM may identify the solution of an optimization problem based on a model which captures only those features of the underlying "real" system that are needed to lead to the right solution, but not necessarily estimate the corresponding optimal performance with accuracy. Even if the exact solution cannot be obtained by such "lower-resolution" models, one can still obtain near-optimal points that exhibit robustness properties with respect to certain aspects of the model they are based on. This property of FM is very promising and deserves further investigation.

Concurrent simulation: One of the accomplishments of this project includes significant breakthroughs in the area of sample derivative estimation for discrete-event systems. In this context, we use a Stochastic Fluid Models (SFM) to approximate the dynamics of the system. Based on this approximation, we derive the "structure" of the sample derivatives of interest which turns out to be unbiased and nonparametric. Finally, we evaluate the sample derivatives from observations on the sample path of the *discrete-event* system (simulated or actual system). Note that several attempts to derive similar sensitivity estimates directly from the discrete-event system are often biased.

The sample derivative analysis is promising and can find applications in several areas such as communication systems, manufacturing systems etc. In the context of communication networks, it is possible to devise extensions to multiple flow classes that can be used for differentiating traffic classes with different Quality-of-Service (QoS) requirements. Ongoing research has already led to very encouraging results, reported in [17, 16, 15], involving IPA estimators and associated optimization for flow control purposes in multi-node models. Furthermore, extensions to networks of SFMs are underway.

Model abstraction using neural networks: The metamodeling procedure we have studied combines simulation of a complex system with the process of training a neural network to become a surrogate model of this system. This exploits the ability of a neural network to act as a universal function approximators. However, if a neural net is to adequately learn the functional relationship between the inputs and outputs of a simulation model it requires a significant number of input/output pairs. Since such information can only be obtained through simulation, it implies that the training phase of the neural network will be rather long. For this reason, we investigate the use of sensitivity information (extracted through some concurrent simulation technique) in the training of neural networks. Our preliminary results indicate that the use of sensitivity information may significantly reduce the number of required input/output training pairs.

The use of sensitivity information during training of neural networks creates several issues that need to be addressed and are part of our future plans. First, the addition of the derivative error in (4.9) makes the training objective function more complex. One problem that has been observed during our experiments is that this addition may create several local minima and as a result convergence issues may arise. Furthermore, the importance associated with the derivative errors (i.e., parameter β) needs to be further investigated. As mentioned earlier, this parameter may be critical to the quality of the approximation as well as the convergence of the training algorithm.

Hierarchical simulation and statistical fidelity: We have investigated the interfacing of high- and low-resolution models in the context of hierarchical simulation. Simple averaging of the output data from a high-resolution simulator to generate input data for a low-resolution simulator is inadequate and occasionally dramatically erroneous. Therefore, to maintain the statistical fidelity, high-resolution output data should be classified into groups that match underlying patterns or features of the system behavior before passing group averages to the low-resolution modules. In an effort to automate the interfacing procedure, we have explored various clustering tools including neural networks and hidden Markov models (HMMs). We demonstrated that HMM is an effective clustering tool especially for problems with high-dimensional input spaces (e.g., sample path clustering).

Sample path clustering can also find applications in areas other than statistical fidelity preservation. For example, we have investigated the HMM in the domain of computer security based on the observation that computer security systems have a hierarchical structure.

Three aspects of behavioral models are discussed with the possible applications of clustering as the important components between different levels of audit events. We then attempted to use HMMs for the purpose of system modelling for anomaly detection. This approach seems promising and we believe that it deserves further investigation.

Appendix A

PROOFS

A.1 Proof of (2.3)

Assume $x_0 = 0$, $y_0 = 0$ and $a_0 = 0$. From Equation (2.2),

$$\begin{aligned}
y_{n+1} &= \max(y_n + a_n - h,\ a_n) \\
&= \max(y_{n-1} + a_n + a_{n-1} - 2h,\ a_n + a_{n-1} - h,\ a_n) \\
&= \vdots \\
&= \max\left\{ \left[y_0 + \sum_{i=0}^{n} a_{n-i} - (n+1)h \right],\ \max_{0 \le j \le n} (\sum_{i=0}^{j} a_{n-i} - jh) \right\} \\
&= \max_{0 \le j \le n} \left(\sum_{i=0}^{j} a_{n-i} - jh \right) \qquad \text{(by } y_0 = 0\text{)}\ . \qquad \text{(A.1)}
\end{aligned}$$

We do not give the induction proof due to the space limitation. Handling Equation (2.1) in the same way, we have

$$x_{n+1} = \max\left[\max_{0 \le j \le n} \left(\sum_{i=0}^{j} a_{n-i} - jh \right) - h,\ 0 \right]\ . \qquad \text{(A.2)}$$

Combining Equation (A.1) and (A.2), we get

$$x_{n+1} = \max(y_{n+1} - h,\ 0)\ . \qquad \text{(A.3)}$$

Given Eqs. (2.1) and (A.3), we can derive

$$\max(y_{n+1} - h,\ 0) = \max(x_n + a_n - h,\ 0), \qquad \forall n > 0\ .$$

So

$$x_n = y_{n+1} - a_n\ ,$$

and

$$Ex = Ey - Ea\ .$$

A.2 Proof of (2.12)

$$\begin{aligned}
F_{re} &= \frac{1}{h}\sum_{k=1}^{h} Ee_k^2 \\
&= \frac{VarX}{h^3}\sum_{k=1}^{h}\sum_{j=1,j\neq k}^{h}\sum_{i=1,i\neq k}^{h}\Big(1+\rho_{|i-j|}-\rho_{|k-i|}-\rho_{|k-j|}\Big) \\
&= \frac{VarX}{h^3}\sum_{k=1}^{h}\left[(h-1)^2+\sum_{j=1,j\neq k,i=j}^{h}\rho_{|i-j|}\right]+\frac{VarX}{h^3}\sum_{k=1}^{h}\sum_{j=1,j\neq k}^{h}\sum_{i=1,i\neq k,i\neq j}^{h}\rho_{|i-j|} \\
&\quad -\frac{VarX}{h^3}\sum_{k=1}^{h}\sum_{j=1,j\neq k}^{h}\sum_{i=1,i\neq k}^{h}\Big(\rho_{|k-i|}+\rho_{|k-j|}\Big) \\
&= F_{un}+B-A\,, \qquad \text{(A.4)}
\end{aligned}$$

where B and A denote the $2nd$ and $3rd$ item in RHS of Equation (A.4) respectively.

$$\begin{aligned}
B &= \frac{Varx}{h^3}\sum_{k=1}^{h}\left[\sum_{j=1}^{h}\left(\sum_{i=1}^{h}\rho_{|i-j|}+\rho_{|k-j|}-1\right)-\left(\sum_{i=1}^{h}\rho_{|i-k|}-2\right)\right] \\
&= \frac{Varx}{h^3}\left[(h-2)\sum_{j=1}^{h}\sum_{i=1}^{h}\rho_{|i-j|}-h(h-2)\right] \\
&= \frac{Varx}{h^3}(h-2)\sum_{j=1}^{h}\sum_{i=1,i\neq j}^{h}\rho_{|i-j|}\ .
\end{aligned}$$

$$\begin{aligned}
A &= \frac{Varx}{h^3}\sum_{k=1}^{h}\left[(h-1)\sum_{i=1,i\neq k}^{h}\rho_{|k-i|}+(h-1)\sum_{j=1,j\neq k}^{h}\rho_{|k-j|}\right] \\
&= \frac{2Varx}{h^3}(h-1)\sum_{j=1}^{h}\sum_{i=1,i\neq j}^{h}\rho_{|i-j|}\ . \qquad \text{(A.5)}
\end{aligned}$$

Let

$$\begin{aligned}
C &= A-B \\
&= 2\frac{Varx}{h^2}\sum_{i=1}^{h-1}(h-i)\rho_i\ . \qquad \text{(A.6)}
\end{aligned}$$

So

$$F_{re}=F_{un}-C\ . \qquad \text{(A.7)}$$

Appendix B

Perturbation Analysis for On-Line Control and Optimization of Stochastic Fluid Models

Christos G. Cassandras, Yorai Wardi, Benjamin Melamed, Gang Sun, Christos G. Panayiotou[1]

To appear in the *IEEE Transactions on Automatic Control*

Abstract

This paper uses Stochastic Fluid Models (SFM) for control and optimization (rather than performance analysis) of communication networks, focusing on problems of buffer control. We derive gradient estimators for packet loss and workload related performance metrics with respect to threshold parameters. These estimators are shown to be unbiased and directly observable from a sample path without any knowledge of underlying stochastic characteristics, including traffic and processing rates (i.e., they are nonparametric). This renders them computable in on-line environments and easily implementable for network management and control. We further demonstrate their use in buffer control problems where our SFM-based estimators are evaluated based on data from an actual system.

[1]Christos G. Cassandras, Gang Sun, and Christos G. Panayiotou are with the Dept. of Manufacturing Engineering, Boston University, Brookline, MA 02446 (cgc@bu.edu,gsun@bu.edu, panayiot@bu.edu). Their work was supported in part by the National Science Foundation under grants EEC-95-27422 and ACI-98-73339, by AFOSR under contract F49620-98-1-0387, by the Air Force Research Laboratory under contract F30602-99-C-0057 and by EPRI/ARO under contract WO8333-03. Yorai Wardi is with the School of Electrical Engineering, Georgia Institute of Technology, Atlanta, GA (wardi@ee.gatech.edu). Benjamin Melamed is with the Dept. of MSIS, Rutgers University, Piscataway, NJ 08854-8054 (melamed@rbs.rutgers.edu). The work of the second and third authors was supported in part by the National Science Foundation under grant DMI-0085659 and by DARPA under contract F30602-00-2-0556.

B.1 Introduction

A natural modeling framework for packet-based communication networks is provided through queueing systems. However, the huge traffic volume that networks are supporting today makes such models highly impractical. It may be impossible, for example, to simulate at the packet level a network slated to transport packets at gigabit-per-second rates. If, on the other hand, we are to resort to analytical techniques from classical queueing theory, we find that traditional traffic models, largely based on Poisson processes, need to be replaced by more sophisticated stochastic processes that capture the bursty nature of realistic traffic; in addition, we need to explicitly model buffer overflow phenomena which typically defy tractable analytical derivations.

An alternative modeling paradigm, based on Stochastic Fluid Models (SFM), has been recently considered for the purpose of analysis and simulation. Introduced in [4] and later proposed in [48] for the analysis of multiplexed data streams and network performance [25], SFMs have been shown to be especially useful for simulating various kinds of high-speed networks [77], [47], [49], [62], [56], [86], [79]. The fluid-flow worldview can provide either approximations to queueing-based models or primary models in their own right. In any event, its justification rests on a molecular view of packets in moderate-to-heavy loads over high-speed transmission links, where the effect of an individual packet or cell on the entire traffic process is virtually infinitesimal, not unlike the effect of a water molecule on the water flow in a river.

The efficacy of a SFM rests on its ability to aggregate multiple events. For example, a discrete event simulation run of an ATM link operating at 622 Megabits-per-second may have to process over a million events per second. On the other hand, if traffic arrives from the source at rates that are piecewise-constant functions of time, then a simulation run would process only one event per rate change. Thus, 30 rate changes per second (as in certain video encoders) may require the processing of only 30 events per second. In effect, the SFM paradigm allows the aggregation of multiple events, associated with the movement of individual packets/cells over a time period of a constant flow rate, into a single event associated with a rate change. It foregoes the identity and dynamics of individual packets and focuses instead on the aggregate flow rate.

For the purpose of performance analysis with Quality of Service (QoS) requirements, the accuracy of SFMs depends on traffic conditions, the structure of the underlying system, and the nature of the performance metrics of interest. By foregoing the identity of individual packets, the SFM paradigm is more suitable for network-related measures, such as buffer levels and packet loss volumes, rather than packet-related measures such as sojourn times (although it is still possible to define fluid-based sojourn times [80]). A QoS metric that depends on the identity of certain packets, for example, cannot be obviously captured by a fluid model. Moreover, some metrics may depend on higher-order statistics of the distributions of the underlying random variables involved, which a fluid model may not be able to accurately capture.

In this paper, our goal is to explore the use of SFMs for the purpose of *control and* *optimization* rather than *performance analysis*. In this case, it is reasonable to expect that the solution of an optimization problem can be identified through a model which captures only those features of the underlying "real" system that are needed to lead to the right solution, even though the corresponding optimal performance may not be accurately estimated. Even if the exact solution cannot be obtained by such "lower-resolution" models, one can still obtain near-optimal points that exhibit robustness with respect to certain aspects of the model they are based on. Such observations have been made in several contexts (e.g., [63]), including recent results related to SFMs reported in [60] where a connection between the SFM and queueing-system-based solution is established for various optimization problems in queueing systems.

With this in mind, we consider here optimization problems for single-node SFMs involving loss volume and workload levels; both are network-related performance metrics associated with buffer control or call-admission control. In a typical buffer control problem, for instance, the optimization problem involves the determination of a threshold (measured in packets or bytes) that minimizes a weighted sum of loss volume and buffer content. As the motivating example presented in Section 2 illustrates, a solution of this problem based on a SFM gives a close approximation to the solution of the associated queueing model. Since solving such problems usually relies on gradient information, estimating the gradient of a given cost function with respect to the aforementioned threshold parameters in a SFM becomes an essential task. Perturbation Analysis (PA) methods [41], [20] are therefore suitable, if appropriately adapted to a SFM viewed as a discrete-event system. Liu and Gong [57], for example, have used PA to analyze an infinite-capacity SFM, with incoming traffic rates as the parameters of interest. In this paper we show that Infinitesimal Perturbation Analysis (IPA) yields remarkably simple sensitivity estimators for packet loss and workload metrics with respect to threshold or buffer size parameters. These estimators also turn out to be *nonparametric* in the sense that they are computable from data directly observable along a sample path, requiring no knowledge of the underlying probability law, including distributions of the random processes involved, or even parameters such as traffic or processing rates. In addition, the estimators obtained are unbiased under very weak structural assumptions on the defining traffic processes. Therefore, the IPA gradient estimators that we derive can be readily used for on-line control purposes to perform periodic network management functions in order to guarantee negotiated QoS parameters and to improve performance. For instance, a network can monitor its relative loss rate and mean buffer contents for a period of time, and then adjust admission parameters, provision transmission capacities, or reassign threshold levels in order to improve performance. Such management functions have not been standardized, and typically are performed in ad-hoc ways by monitoring performance levels. Aside from solving explicit optimization problems, IPA gradient estimators simplify the implementation of sensitivity analysis.

The contributions of this paper are as follows. First, we consider a single-node SFM and derive IPA gradient estimators for performance metrics related to loss and workload levels with respect to threshold parameters (equivalently, buffer sizes). One can derive such estimators by either (a) considering the finite difference of a performance metric as a function of the finite difference of a parameter and then use explicit limit arguments to obtain an unbiased estimate of the performance metric derivative, or (b) deriving the sample derivative for the performance metric involved, and then proving that it indeed yields an unbiased estimate. The former approach provides clear insights into the dynamic process of generation and propagation of perturbations, which is very helpful in understanding how to extend the approach to multiple fluid classes and multiple nodes. In addition, it requires no technical conditions on the traffic processes or the sample functions involved. However, this approach is tedious, even for a simple single-node model. The latter approach is simpler and more elegant, at the expense of some mild technical conditions needed to justify the evaluation of the sample derivative. It requires, however, some results from the first approach in order to prove unbiasedness of the derived estimators. Thus, in this paper, we start with the former approach, and then show that the estimators derived are equivalent to the latter, which we subsequently adopt. Based on these estimators, we also present simple algorithms for implementing them on line, taking advantage of their nonparametric nature.

The second contribution of the paper is to make use of the IPA gradient estimators derived to tackle buffer control as an optimization problem. In particular, we seek to determine the threshold value that minimizes a given performance metric. Packet-by-packet buffer control can be applied after the session admission decision is made in order to dynamically adjust network resources so as to minimize some cost based on the promised QoS. We use a standard gradient-based stochastic optimization scheme, where we estimate the gradient of the performance function with respect to the threshold parameter on the SFM; however, due to the simplicity of this gradient estimator, we evaluate it *based on data observed on a sample path of the actual (discrete-event) system.* Thus, we use the SFM only to obtain a gradient estimator; the associated value at any operating point is obtained from real system data.

The paper is organized as follows. First, in Section 2, we motivate our approach with a buffer control problem in the SFM setting and show the application of IPA to it. In Section 3, we describe the detailed SFM setting and define the performance metrics and parameters of interest. In Section 4, we derive IPA estimators for the sensitivities of the expected loss rate and workload with respect to threshold parameters (equivalently, buffer sizes) and show their unbiasedness. This is first demonstrated by a direct approach based on finite differences. The IPA approach is then generalized by evaluating sample derivatives (at the expense of introducing some mild technical conditions); these are shown to provide unbiased performance derivative estimators which are of nonparametric nature. Algorithms for implementing the derivative estimators obtained are also provided. In Section 5, we show how the SFM-based derivative estimates can be used on line using data from the *actual* system (not the SFM) in order to solve buffer control problems. Finally, in Section 6 we outline a number of open problems and future research directions motivated by this work.

B.2 A Motivating Example: Threshold-Based Buffer Control

This section presents a motivating example of buffer control in the setting of both a queueing model and a SFM and then compares the two. Consider a network node where buffer control

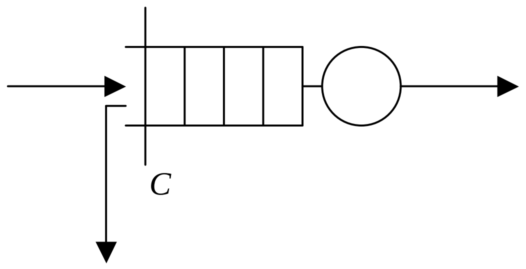


Figure B.1: Buffer control in a single node

at the packet level takes place using a simple threshold-based policy: when a packet arrives and the queue length is below a given amount C, it is accepted; otherwise it is rejected. Let $\bar{L}(C)$ denote the expected loss rate, i.e., the expected rate of packet overflow at steady state, and let $\bar{Q}(C)$ denote the mean queue length when the threshold is C. We then define the cost function

$$J(C) \ = \ \bar{Q}(C) \ + \ R \cdot \bar{L}(C), \tag{B.1}$$

where R is a penalty associated with rejecting a packet. Thus, $J(C)$ captures the tradeoff between providing satisfactory service (low delay) and rejecting too many packets. Since, arguably, the notion of steady state is hard to justify in many networks, and since control decisions need to be made periodically or in response to apparent adverse network conditions, a more realistic performance measure is one where $\bar{L}(C)$ and $\bar{Q}(C)$ are replaced by $\bar{L}_T(C)$ and $\bar{Q}_T(C)$, the expected loss rate and mean queue length, respectively, over the time-interval $[0, T]$. We then consider

$$J_T(C) \ = \ \bar{Q}_T(C) \ + R \cdot \bar{L}_T(C) \tag{B.2}$$

to be the cost function of interest. Care must be taken in defining the above expectations over a finite time-horizon, since they generally depend on the initial conditions; for the time being, we shall assume that the queue is empty at time $t = 0$, and revisit this point later. Figure B.1 depicts the queueing system under consideration.

The packet arrival process is modeled as an ON-OFF source so that packets arrive at a peak rate α during an ON period, followed by an OFF period during which no packets arrive. The packet processing rate is β. For the example used here and illustrated in Fig. B.1, the number of arrivals in each ON period is geometrically distributed with parameter $p = 0.05$ and arrival rate $\alpha = 2$; the OFF period is exponentially distributed with parameter $\mu = 0.1$; and the service rate is $\beta = 1.01$. Thus, the traffic intensity of the system is $\alpha(\frac{1}{\alpha p})/\beta(\frac{1}{\alpha p} + \frac{1}{\mu}) = 0.99$, where $\frac{1}{\alpha p}$ is the average length of an ON period and $\frac{1}{\mu}$ is the average length of an OFF period. The cost function $J_T(C)$ in this problem is piecewise constant, hence gradient-based algorithms cannot be used. However, by exhaustively simulating this queueing system and averaging over 25 sample paths of length $T = 100,000$ time units and estimating $J_T(C)$ over different discrete values of C, we obtained the curve labeled 'DES' in Fig. B.2, using a rejection penalty $R = 50$. One can see that the optimal threshold value in this example is $C^* = 15$.

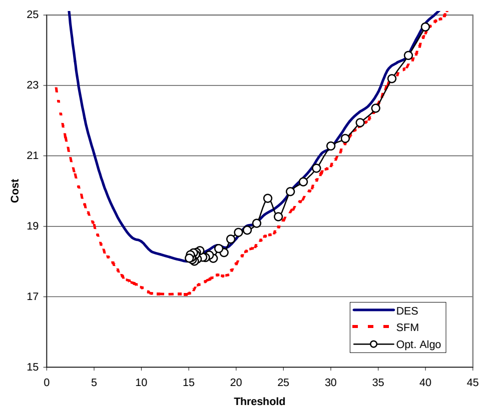


Figure B.2: Cost v. threshold comparison for DES and SFM

Next, we adopt a simple SFM for the same system, treating packets as "fluid". During an ON period, the fluid volume in the buffer, $x(t)$, increases at rate $\alpha - \beta$ (we assume $\alpha > \beta$, otherwise there would be no buffer accumulation), while during an OFF period it decreases at a rate β. The cost function in this model is

$$J_T^{SFM}(\theta) = \bar{Q}_T^{SFM}(\theta) + R \cdot \bar{L}_T^{SFM}(\theta) \tag{B.3}$$

where $\theta \in \mathbb{R}^+$ is the threshold used to reject incoming fluid when the buffer fluid volume reaches level θ. The corresponding expected loss rate and mean buffer fluid volume over the time-interval $[0, T]$ are denoted by $\bar{L}_T^{SFM}(\theta)$ and $\bar{Q}_T^{SFM}(\theta)$, respectively. Simulating this model under the same ON-OFF conditions as before over many values of θ results in the curve labeled "SFM" in Fig. B.2. The important observation is that the two optima are close, whereas the difference in the actual cost estimates can be substantial (especially for a lightly loaded system). In fact, $\theta^* = 13$ and $J_T^{SFM}(13) = 17.073$, as compared to $J_T(13) = 18.127$ and the optimal $J_T(C^*) = J_T(15) = 18.012$.

Based on this observation, we are motivated to study means for efficiently identifying solutions to problems formulated in a SFM setting. It is still difficult to obtain analytical solutions, however, since expressions for $\bar{Q}_T^{SFM}(\theta)$ and $\bar{L}_T^{SFM}(\theta)$ are unavailable, unless the arrival and service processes in the actual system are very simple. Therefore, one needs to resort to iterative methods such as stochastic approximation algorithms (e.g., [50]), which are driven by estimates of the gradient of a cost function with respect to the parameters of interest.

In the case of the simple buffer control problem above, we are interested in estimating $dJ_T/d\theta$ based on directly observed (simulated) data. We can then seek to obtain θ^* such that it minimizes $J_T(\theta)$ through an iterative scheme of the form

$$\theta_{n+1} = \theta_n - \nu_n H_n(\theta_n, \omega_n^{SFM}), \qquad n = 0, 1, \ldots \tag{B.4}$$

where $\{\nu_n\}$ is a step size sequence and $H_n(\theta_n, \omega_n^{SFM})$ is an estimate of $dJ_T/d\theta$ evaluated at $\theta = \theta_n$ and based on information obtained from a sample path of the SFM denoted by

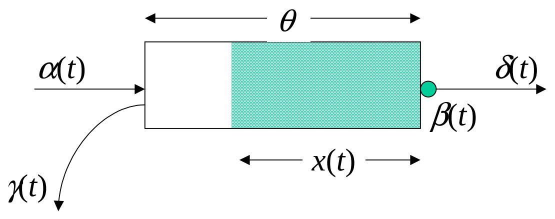


Figure B.3: The basic Stochastic Fluid Model (SFM)

ω_n^{SFM}. However, as we will see, the simple form of $H_n(\theta_n, \omega_n^{SFM})$ to be derived also enables us to apply the same scheme to the original discrete event system:

$$C_{n+1} = C_n - \nu_n H_n(C_n, \omega_n^{DES}), \qquad n = 0, 1, \ldots \tag{B.5}$$

where C_n is the threshold used for the nth iteration and ω_n^{DES} is a sample path of the discrete event system. In other words, analyzing the SFM provides us with the *structure* of a gradient estimator whose actual value can be obtained based on data from the actual system. In Fig. B.2, the curve labeled "Opt.Algo." corresponds to this process and illustrates how one can indeed recover the optimal threshold $C^* = 15$.

B.3 The Stochastic Fluid Model (SFM) Setting

The SFM setting is based on the fluid-flow worldview, where "liquid molecules" flow in a continuous fashion. The basic SFM, used in [80] and shown in Figure 3, consists of a single-server (spigot) preceded by a buffer (fluid storage tank), and it is characterized by five stochastic processes, all defined on a common probability space $(\Omega, \mathcal{F}, P)$ as follows:

- $\{\alpha(t)\}$: the input flow (inflow) rate to the SFM,
- $\{\beta(t)\}$: the service rate, i.e., the maximal fluid discharge rate from the server,
- $\{\delta(t)\}$: the output flow (outflow) rate from the SFM, i.e., the actual fluid discharge rate from the server,
- $\{x(t)\}$: the buffer occupancy or buffer content, i.e., the volume of fluid in the buffer,
- $\{\gamma(t)\}$: the overflow (spillover) rate due to excessive incoming fluid at a full buffer.

The above processes evolve over a time interval $[0, T]$ for a given fixed $T > 0$. The inflow process $\{\alpha(t)\}$ and the service-rate process $\{\beta(t)\}$ are assumed to be right-continuous piecewise constant, with $0 \leq \alpha_{\min} \leq \alpha(t) \leq \alpha_{\max} < \infty$ and $0 \leq \beta_{\min} \leq \beta(t) \leq \beta_{\max} < \infty$. Let θ denote the size of the buffer, which is the variable parameter we will concentrate on for the purpose of IPA. The processes $\{\alpha(t)\}$ and $\{\beta(t)\}$, along with the buffer size θ, define the behavior of the SFM. In particular, they determine the buffer content, $x(\theta; t)$, the overflow rate $\gamma(\theta; t)$, and the output flow $\delta(\theta; t)$. The notational dependence on θ indicates that we will analyze performance metrics as functions of the given θ. We will assume that the real-valued parameter θ is confined to a closed and bounded (compact) interval Θ; to avoid unnecessary technical complications, we assume that $\theta > 0$ for all $\theta \in \Theta$.

The buffer content $x(\theta;t)$ is determined by the following one-sided differential equation,

$$\frac{dx(\theta;t)}{dt^+} = \begin{cases} 0, & \text{if } x(\theta;t) = 0 \text{ and } \alpha(t) - \beta(t) \leq 0, \\ 0, & \text{if } x(\theta;t) = \theta \text{ and } \alpha(t) - \beta(t) \geq 0, \\ \alpha(t) - \beta(t), & \text{otherwise} \end{cases} \tag{B.6}$$

with the initial condition $x(\theta;0) = x_0$ for some given x_0; for simplicity, we set $x_0 = 0$ throughout the paper. The outflow rate $\delta(\theta;t)$ is given by

$$\delta(\theta;t) \ = \ \begin{cases} \beta(t), & \text{if } x(\theta;t) > 0, \\ \alpha(t), & \text{if } x(\theta;t) = 0, \end{cases} \tag{B.7}$$

where we point out that if we allow $\theta = 0$, then $\delta(\theta;t) \ = \min\{\alpha(t), \beta(t)\}$. The overflow rate $\gamma(\theta;t)$ is given by

$$\gamma(\theta;t) \ = \ \begin{cases} \max\{\alpha(t) - \beta(t), 0\}, & \text{if } x(\theta;t) = \theta, \\ 0, & \text{if } x(\theta;t) < \theta. \end{cases} \tag{B.8}$$

This SFM can be viewed as a dynamic system whose input consists of the two *defining* processes $\{\alpha(t)\}$ and $\{\beta(t)\}$ along with the buffer size θ, its state is comprised of the buffer content process, and its output includes the outflow and overflow processes. The state and output processes are referred to as *derived* processes, since they are determined by the defining processes. Since the input sample functions (realizations) of $\{\alpha(t)\}$ and $\{\beta(t)\}$ are piecewise constant and right-continuous, the state trajectory $x(\theta;t)$ is piecewise linear and continuous in t, and the output function $\gamma(\theta;t)$ is piecewise constant. Moreover, the state trajectory can be decomposed into two kinds of intervals: *empty periods* and *busy periods*. Empty Periods (EP) are maximal intervals during which the buffer is empty, while Busy Periods (BP) are supremal intervals during which the buffer is nonempty. Observe that during an EP the system is not necessarily idle since the server may be active; see (B.7). Note also that since $x(\theta;t)$ is continuous in t, EPs are always closed intervals, whereas BPs are open intervals unless containing one of the end points 0 or T. The outflow process $\{\delta(t)\}$ becomes important in modeling networks of SFMs and it will not concern us any further here, since our interest in this paper lies in single-node systems.

Let $\mathcal{L}(\theta) : \Theta \to \mathbb{R}$ be a random function defined over the underlying probability space $(\Omega, \mathcal{F}, P)$. Strictly speaking, we write $\mathcal{L}(\theta, \omega)$ to indicate that this sample function depends on the sample point $\omega \in \Omega$, but will suppress ω unless it is necessary to stress this fact. In what follows, we will consider two performance metrics, the *Loss Volume* $L_T(\theta)$ and the *Cumulative Workload* (or just *Work*) $Q_T(\theta)$, both defined on the interval $[0,T]$ via the following equations:

$$L_T(\theta) \ = \ \int_0^T \gamma(\theta;t)dt, \tag{B.9}$$

$$Q_T(\theta) \ = \ \int_0^T x(\theta;t)dt, \tag{B.10}$$

where, as already mentioned, we assume that $x(\theta;0)=0$. Observe that $\frac{1}{T}E\left[L_T(\theta)\right]$ is the *Expected Loss Rate* over the interval $[0,T]$, a common performance metric of interest (from which related metrics such as *Loss Probability* can also be derived). Similarly, $\frac{1}{T}E\left[Q_T(\theta)\right]$ is the *Expected Buffer Content* over $[0,T]$. We may then formulate optimization problems such as the determination of θ^* that minimizes a cost function of the form

$$J(\theta)=\frac{1}{T}E\left[Q_T(\theta)\right]+\frac{R}{T}E\left[L_T(\theta)\right]\equiv\frac{1}{T}J_Q(\theta)+\frac{R}{T}J_L(\theta),$$

where R represents a rejection cost due to overflow. In order to accomplish this task, we rely on estimates of $dJ_L(\theta)/d\theta$ and $dJ_Q(\theta)/d\theta$ provided by the sample derivatives $dL_T(\theta)/d\theta$ and $dQ_T(\theta)/d\theta$ for use in stochastic gradient-based schemes. Accordingly, the objective of the next section is the estimation of the derivatives of $J_L(\theta)$ and $J_Q(\theta)$, which we will pursue through Infinitesimal Perturbation Analysis (IPA) techniques [41], [20]). Henceforth we shall use the "prime" notation to denote derivatives with respect to θ, and will proceed to estimate the derivatives $J_L^{'}(\theta)$ and $J_Q^{'}(\theta)$. The corresponding sample derivatives are denoted by $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$, respectively.

B.4 Infinitesimal Perturbation Analysis (IPA) with respect to Buffer Size or Threshold

As already mentioned, we will concentrate on the buffer size θ in the SFM described above or, equivalently, a threshold parameter used for buffer control. We assume that the processes $\{\alpha(t)\}$ and $\{\beta(t)\}$ are independent of θ and of the buffer content. Thus, we consider network settings operating with protocols such as ATM and UDP, but not TCP. Our objective is to estimate the derivatives $J_L^{'}(\theta)$ and $J_Q^{'}(\theta)$ through the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ which are commonly referred to as Infinitesimal Perturbation Analysis (IPA) estimators; comprehensive discussions of IPA and its applications can be found in [41], [20]. The IPA derivative-estimation technique computes $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ along an observed sample path ω. An IPA-based estimate $\mathcal{L}'(\theta)$ of a performance metric derivative $dE[\mathcal{L}(\theta)]/d\theta$ is *unbiased* if $dE[\mathcal{L}(\theta)]/d\theta=E[\mathcal{L}'(\theta)]$. Unbiasedness is the principal condition for making the application of IPA practical, since it enables the use of the sample (IPA) derivative in control and optimization methods that employ stochastic gradient-based techniques.

We consider sample paths of the SFM over $[0,T]$. For a fixed $\theta\in\Theta$, the interval $[0,T]$ is divided into alternating EPs and BPs. Suppose there are K busy periods denoted by $\mathcal{B}_k$, $k=1,\ldots,K$, in increasing order. Then, by (B.9)-(B.10), the sample performance functions assume the following form:

$$L_T(\theta) = \sum_{k=1}^{K}\int_{\mathcal{B}_k}\gamma(\theta;t)dt, \tag{B.11}$$

$$Q_T(\theta) = \sum_{k=1}^{K}\int_{\mathcal{B}_k}x(\theta;t)dt. \tag{B.12}$$

As mentioned earlier, the processes $\{\alpha(t)\}$ and $\{\beta(t)\}$ are assumed piecewise constant. This implies that, w.p.1, there exist a random integer $N(T) > 0$ and an increasing sequence of time points $0 = t_0 < t_1 < \ldots < t_{N(T)} < t_{N(T)+1} = T$, generally dependent upon the sample path ω, such that t_i is a jump (discontinuity) point of $\alpha(t) - \beta(t)$; clearly, $\alpha(t) - \beta(t)$ is continuous at all points other than $t_0, \ldots, t_{N(T)}$. We will assume that $N(T)$ has a finite expectation, i.e., $E[N(T)] < \infty$.

Viewed as a discrete-event system, an *event* in a sample path of the SFM may be either *exogenous* or *endogenous*. An exogenous event is a jump in either $\{\alpha(t)\}$ or $\{\beta(t)\}$. An endogenous event is defined to occur when the buffer becomes full or empty. We note that the times at which the buffer *ceases to be* full or empty are locally independent of θ, because they correspond to a change of sign in the difference function $\alpha(t) - \beta(t)$ (by a random function $f(\theta)$ being "locally independent" of θ we mean that for a given θ there exists $\Delta\theta > 0$ such that for every $\bar{\theta} \in (\theta - \Delta\theta, \theta + \Delta\theta)$, w.p.1 $f(\bar{\theta}) = f(\theta)$, where $\Delta\theta$ may depend on both θ and on the sample path). Thus, given a BP $\mathcal{B}_k$, its starting point is one where the buffer ceases to be empty and is therefore locally independent of θ, while its end point generally depends on θ. Denoting these points by ξ_k and $\eta_k(\theta)$ we express $\mathcal{B}_k$ as

$$\mathcal{B}_k \;=\; (\xi_k, \eta_k(\theta)), \qquad k = 1, \ldots, K$$

for some random integer K. The BPs can be classified according to whether some overflow occurs during them or not. Thus, we define the random set

$$\begin{aligned}\Phi(\theta) \;:=\; \{k \in \{1, \ldots, K\} :\; & x(t) = \theta, \\ & \alpha(t) - \beta(t) > 0 \text{ for some } t \in (\xi_k, \eta_k(\theta))\}.\end{aligned}$$

For every $k \in \Phi(\theta)$, there is a (random) number $M_k \geq 1$ of *overflow periods* in $\mathcal{B}_k$, i.e., intervals during which the buffer is full and $\alpha(t) - \beta(t) > 0$. Let us denote these overflow periods by $\mathcal{F}_{k,m}$, $m = 1, \ldots, M_k$, in increasing order and express them as $\mathcal{F}_{k,m} = [u_{k,m}(\theta), v_{k,m}]$, $\quad k = 1, \ldots, K$. Observe that the starting time $u_{k,m}(\theta)$ generally depends on θ, whereas the ending time $v_{k,m}$ is locally independent of θ, since it corresponds to a change of sign in the difference function $\alpha(t) - \beta(t)$, which has been assumed independent of θ. Finally let

$$B(\theta) = |\Phi(\theta)| \tag{B.13}$$

where $|\cdot|$ denotes the cardinality of a set, i.e., $B(\theta)$ is the number of BPs in $[0, T]$ during which some overflow is observed. To summarize:

- There are K busy periods in $[0, T]$, with $\mathcal{B}_k = (\xi_k, \eta_k(\theta)), \;\; k = 1, \ldots, K$.
- $k \in \Phi(\theta)$ iff some overflow occurs during $\mathcal{B}_k$; we set $B(\theta) = |\Phi(\theta)|$.
- For each $k \in \Phi(\theta)$, there are M_k overflow periods in $\mathcal{B}_k$, i.e., $\mathcal{F}_{k,m} = [u_{k,m}(\theta), v_{k,m}]$, $m = 1, \ldots, M_k$.

A typical sample path is shown in Fig. B.4, where $K = 3$, $\Phi = \{1, 3\}$, $M_1 = 2$, $M_2 = 0$, $M_3 = 1$.

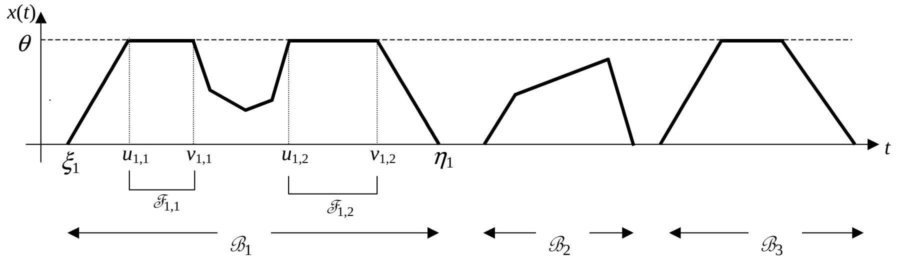


Figure B.4: A typical sample path of a SFM

As mentioned in the Introduction, we present two ways of deriving IPA estimators: (i) by evaluating the finite differences $\Delta L_T(\theta)$ and $\Delta Q_T(\theta)$ as functions of $\Delta\theta$, obtaining left and/or right sample derivatives (depending on whether $\Delta\theta < 0$ or $\Delta\theta > 0$), taking limits as $\Delta\theta \to 0$, and finally exploring if they yield unbiased estimates of $J_L^{'}(\theta)$ and $J_Q^{'}(\theta)$; or (ii) by explicitly evaluating $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$, which requires some additional technical assumptions. We will first proceed with the former approach and consider only the loss volume metric $L_T(\theta)$; the analysis for $Q_T(\theta)$ is similar, though a bit more involved (see also [15]). In pursuing this approach, we will also derive some results that will be used to establish the unbiasedness of the estimators $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ obtained through the latter approach.

B.4.1 IPA Using Finite Difference Analysis

The stochastic component of the SFM manifests itself in the duration of the intervals defined by exogenous event occurrences corresponding to jumps in either $\alpha(t)$ or $\beta(t)$. Let $\{A_i\}$, $i = 1, 2, \ldots$, be the point process defined by these exogenous event times. For convenience, let α_i and β_i denote the (constant) inflow rate and service rate, respectively, over the interval $[A_i, A_{i+1})$. Note that we do not impose any restrictions on the probability law of the intervals defined by these events.

The main result of this section is to show that the sample derivative $L_T^{'}(\theta)$, i.e., the sensitivity of the loss volume with respect to θ, is given by $-B(\theta)$, and that this is an unbiased estimator of $J_L^{'}(\theta)$. Recall that $B(\theta)$ is simply the count of busy periods in which at least one overflow period is observed. Moreover, this remarkably simple estimator is independent of any assumptions on the traffic process or service process, as well as of the rates involved and even θ, i.e., it is *nonparametric.*

The starting point in IPA is to consider a *nominal* sample path under some buffer size (equivalently, admission threshold) θ and a *perturbed* sample path resulting from perturbing θ by $\Delta\theta$, while keeping the realizations of the processeses $\{\alpha(t)\}$ and $\{\beta(t)\}$ unchanged, hence leaving $\{A_i\}$, $i = 1, 2, \ldots$, unchanged. For simplicity, we limit ourselves to the case where $\Delta\theta > 0$, leading to an estimate of the right sample derivative of $L_T(\theta)$; the case where $\Delta\theta < 0$ is similar, leading to an estimate of the left sample derivative of $L_T(\theta)$. We then define

$$\Delta x_i(\theta, \Delta\theta) = x_i(\theta + \Delta\theta) - x_i(\theta),$$

where $x_i(\theta)$ denotes the nominal sample buffer content at time A_i and $x_i(\theta + \Delta\theta)$ denotes the perturbed sample buffer content at the same time. Similarly, we define perturbations for some additional sample path quantities as follows. First, setting $A_0 = 0$, let

$$L_i(\theta) = \int_{A_{i-1}}^{A_i} \gamma(\theta; t)dt, \quad i = 1, 2, \ldots \tag{B.14}$$

be the total loss volume observed over an interevent interval $[A_{i-1}, A_i)$, and define

$$\Delta L_i(\theta, \Delta\theta) = L_i(\theta + \Delta\theta) - L_i(\theta) \tag{B.15}$$

In addition, let

$$y_{i+1}(\theta) = x_i(\theta) + (\alpha_i - \beta_i)[A_{i+1} - A_i] \tag{B.16}$$

and note that $(\alpha_i - \beta_i)[A_{i+1} - A_i]$ is simply the amount of change in the buffer content from time A_i to time A_{i+1}. Therefore, $y_{i+1}(\theta)$ is the queue content obtained at time A_{i+1} if the queue were allowed to become negative or to exceed θ. We may then define

$$\Delta y_i(\theta, \Delta\theta) = y_i(\theta + \Delta\theta) - y_i(\theta)$$

Finally, we define a perturbation in the ending time of a BP as

$$\Delta\eta_k(\theta, \Delta\theta) = \eta_k(\theta + \Delta\theta) - \eta_k(\theta), \quad k = 1, 2, \ldots$$

For notational simplicity, we shall henceforth suppress the arguments of all quantities Δx_i, Δy_i, ΔL_i, $\Delta\eta_k$.

Consider a typical BP, $\mathcal{B}_k$, and all possible events that can take place in it, so as to determine how associated perturbations are either generated (due to $\Delta\theta$) or propagated from the previous event. The kth busy period is initiated by an exogenous event at time $\xi_k = A_i$, for some i, such that $\alpha_i - \beta_i > 0$, and let us assume that $\Delta x_i = 0$. Regarding the next exogenous event at time A_{i+1} there are two possible cases to consider:

Case I: $y_{i+1}(\theta) \leq \theta$. In this case, $y_{i+1}(\theta)$ is given by (B.16) and we have (see also Fig. B.5(a)):

$$x_{i+1}(\theta) = y_{i+1}(\theta)$$
$$L_i(\theta) = 0$$

Clearly, $\Delta x_{i+1} = \Delta y_{i+1} = \Delta L_{i+1} = 0$.

Case II: $y_{i+1}(\theta) > \theta$. In this case, the queue content in the perturbed path can increase beyond θ up to the perturbed value $\theta + \Delta\theta$. Then, as also seen in Fig. B.5(b),

$$\Delta x_{i+1} = \Delta\theta \tag{B.17}$$
$$\Delta L_{i+1} = -\Delta\theta \tag{B.18}$$

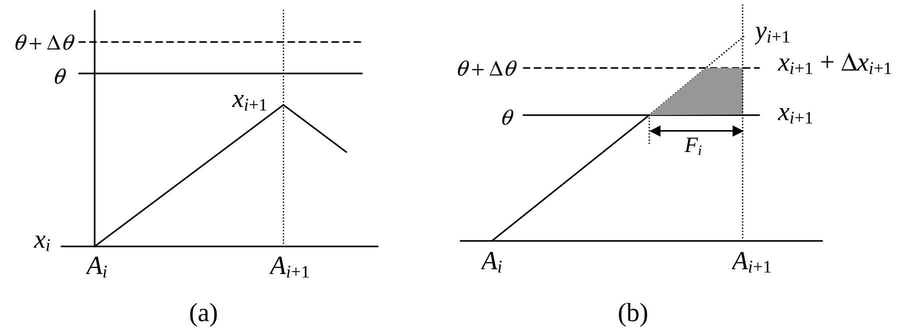


Figure B.5: (a) Case I: No perturbation generation ($y_{i+1}(\theta) \le \theta$). (b) Case II: Perturbation generation for $0 < \Delta\theta \le y_{i+1}(\theta) - \theta$

provided that $\Delta\theta$ is such that $0 < \Delta\theta \le y_{i+1}(\theta) - \theta$. To consider the case where $\Delta\theta > y_{i+1}(\theta) - \theta$, let the length of the overflow period in the nominal path be F_i and note that

$$F_i = \frac{y_{i+1} - \theta}{\alpha_i - \beta_i}.$$

Thus, if $\Delta\theta > y_{i+1}(\theta) - \theta = (\alpha_i - \beta_i)F_i$, then it is easy to see that the shaded area in Fig. B.5(b) reduces to a triangle with area $\frac{1}{2}(\alpha_i - \beta_i)F_i^2$. We then get

$$\Delta x_{i+1} = (\alpha_i - \beta_i)F_i \tag{B.19}$$
$$\Delta L_{i+1} = -(\alpha_i - \beta_i)F_i \tag{B.20}$$

Using the standard notation $[x]^+ = \max(x, 0)$, we can combine (B.17)-(B.18) with (B.19)-(B.20) to write

$$\Delta x_{i+1} = \Delta\theta - [\Delta\theta - (\alpha_i - \beta_i)F_i]^+ \tag{B.21}$$
$$\Delta L_{i+1} = -\Delta\theta + [\Delta\theta - (\alpha_i - \beta_i)F_i]^+ \tag{B.22}$$

Equations (B.21)-(B.22) capture the perturbation *generation* process due to $\Delta\theta$. The next step is to study how perturbations can be *propagated*, assuming the general situation $\Delta x_i \ge 0$. Doing so leads to the following result, which describes the complete queue content perturbation dynamics and establishes bounds for Δx_i.

Lemma 4 *For all $i = 1, 2, \ldots,$*

$$0 \le \Delta x_i \le \Delta\theta \tag{B.23}$$

and

$$\Delta x_{i+1} = \begin{cases} [\Delta x_i - (\beta_i - \alpha_i)I_i]^+ & \text{if } \alpha_i - \beta_i < 0 \\ \Delta\theta - [\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i]^+ & \text{if } \alpha_i - \beta_i \ge 0 \end{cases} \tag{B.24}$$

where I_i is the length of an EP ending at A_{i+1} with $I_i = 0$ if no such period exists, and F_i is the length of an overflow period ending at A_{i+1} with $F_i = 0$ if no such period exists.

Proof. See Appendix.

An immediate consequence of Lemma 4 is that a queue content perturbation may propagate across busy periods depending on the length of the EP separating these busy periods. This is because $\Delta x_{i+1} = [\Delta x_i - (\beta_i - \alpha_i)I_i]^+ \geq 0$ when an event occurs at time A_{i+1} that ends an EP of length I_i. Moreover, recalling that the endpoints of busy periods are denoted by $\eta_k(\theta)$, $k = 1, 2, \ldots$, the perturbation in $\eta_k(\theta)$ can be easily obtained by noticing in Fig. B.8(a) (**Case 1.2** in the proof of Lemma 4) that

$$\Delta\eta_k(\theta) = \frac{\Delta\theta}{\beta_i - \alpha_i}, \tag{B.25}$$

provided that $\Delta\theta \leq (\beta_i - \alpha_i)I_i$, where α_i and β_i are the inflow rate and service rate at the time the BP ends. To account for the fact that the kth BP may contain an overflow interval of length F_i with $\Delta\theta > (\alpha_i - \beta_i)F_i + \Delta x_i$, $\Delta\theta$ in (B.25) can be replaced by $\Delta\theta - [\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i]^+ \leq \Delta\theta$ in view of (B.24). If, on the other hand, $\Delta\theta > (\beta_i - \alpha_i)I_i$, then the kth and $(k+1)$th busy periods are merged, which implies that $\Delta\eta_k(\theta)$ includes the entire length of the $(k+1)$th busy period.

Next, we identify bounds for ΔL_i (a generalization of the bounds for Δx_i and ΔL_i can also be found in [80]).

Lemma 5 *For all $i = 1, 2, \ldots,$*

$$-\Delta\theta \leq \Delta L_i \leq 0 \tag{B.26}$$

Proof. See Appendix.

Recall that if at least one overflow period is observed in the kth BP, then $k \in \Phi(\theta)$. Making use of the standard indicator function $\mathbf{1}[k \in \Phi(\theta)] = 1$ if $k \in \Phi(\theta)$ and zero otherwise, we have the following result, which allows us to characterize the cumulative loss perturbation at the end of a BP, which we will denote by $\Lambda_k(\Delta\theta)$, $k = 1, \ldots, K$.

Lemma 6 *Consider a BP $\mathcal{B}_k = (\xi_k, \eta_k(\theta))$ with $\xi_k = A_j$, $\Delta x_j = 0$, and $A_m < \eta_k(\theta) \leq A_{m+1}$. Assuming $\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i \leq 0$ for all $i = j, \ldots, m$, the cumulative loss perturbation at the end of this busy period is*

$$\Lambda_k(\Delta\theta) = -\Delta\theta\mathbf{1}[k \in \Phi(\theta)], \qquad k = 1, \ldots, K \tag{B.27}$$

Proof. See Appendix.

In simple terms, the loss perturbation depends *only* on the presence of an overflow within the observed busy period and not their number. It is noteworthy that this perturbation does not explicitly depend on any values that $\alpha(t)$ or $\beta(t)$ may take or the nature of the stochastic processes involved. Considering Lemma 6, note that it allows us to analyze all busy periods separately and accumulate loss perturbations at the end of the sample path over all busy periods observed; this, however, is contingent on the fact that $\Delta x_i = 0$ when a BP starts with an exogenous event at A_i. On the other hand, we saw that a consequence of Lemma 4 is $\Delta x_{i+1} = [\Delta x_i - (\beta_i - \alpha_i)I_i]^+$ following an EP of length I_i, i.e., the buffer content perturbation may not be zero when a BP starts, depending on the length of the EP separating it from the preceding BP.

We can now derive an unbiased derivative estimate for our performance metric by establishing the following result.

Theorem 7 *The (right) derivative of the Expected Loss, $E[L_T(u)]$, is given by*

$$\frac{dE[L_T(\theta)]}{d\theta} = -E\left[\sum_{k=1}^{K} \mathbf{1}[k \in \Phi(\theta)]\right] = -E\left[B(\theta)\right] \tag{B.28}$$

where K is the (random) number of busy periods contained in $[0,T]$, including a possibly incomplete last busy period.

Proof. We have

$$\begin{aligned}\frac{dE[L_T(\theta)]}{d\theta} &= \lim_{\Delta\theta\to 0} \frac{1}{\Delta\theta} E\left[\Delta L_T(\theta)\right] \\ &= \lim_{\Delta\theta\to 0} \frac{1}{\Delta\theta} E\left[\sum_{k=1}^{K} \Lambda_k(\Delta\theta)\right]\end{aligned}$$

where $\Lambda_k(\Delta\theta) = -\Delta\theta\mathbf{1}[k \in \Phi(\theta)]$ from Lemma 6, provided $\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i \leq 0$ for all $A_i \in [0,T]$. It follows that

$$\frac{dE[L_T(\theta)]}{d\theta} = -E\left[\sum_{k=1}^{K} \mathbf{1}[k \in \Phi(\theta)]\right] = -E\left[B(\theta)\right]$$

where we have used the definition in (B.13).

If $\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i > 0$ for some $A_i \in [0,T]$, then the only additional effect comes from $\Delta L_{i+1} = -(\alpha_i - \beta_i)F_i < 0$ in (B.53). Then, consider

$$\begin{aligned}&E[-(\alpha_i - \beta_i)F_i \mid \Delta\theta - \Delta x_i > (\alpha_i - \beta_i)F_i] \\ &\qquad = \int_0^{\Delta\theta - \Delta x_i} -xf(x)dx\end{aligned}$$

where $f(\cdot)$ is the conditional pdf of $(\alpha_i - \beta_i)F_i$ given $\Delta\theta - \Delta x_i > (\alpha_i - \beta_i)F_i$, and let $f(\cdot) \leq c < \infty$. Recalling that $0 \leq \Delta x_i \leq \Delta\theta$ from Lemma 4, we get

$$\begin{aligned}\int_0^{\Delta\theta - \Delta x_i} -xf(x)dx &\geq \int_0^{\Delta\theta} -xf(x)dx \\ &\geq \int_0^{\Delta\theta} -\Delta\theta f(x)dx \\ &\geq \Delta\theta \int_0^{\Delta\theta} -cdx = -c\left(\Delta\theta\right)^2\end{aligned}$$

and it follows that

$$E\left[-(\alpha_i - \beta_i)F_i\right] \geq -c\left(\Delta\theta\right)^2 \tag{B.29}$$

The cumulative loss perturbation due to events such that $\Delta L_{i+1} = -(\alpha_i - \beta_i)F_i$ is bounded from below by

$$\sum_{i=1}^{N(T)} -(\alpha_i - \beta_i)F_i,$$

where F_i is the length of an overflow interval after the ith exogenous event, with $F_i = 0$ if no such overflow interval is present, and $N(T)$ is the total number of exogenous events in $[0, T]$. This cumulative loss perturbation is also bounded from above by 0, since $\Delta L_i \leq 0$ from Lemma 5. Using (B.29), we get, given some $N(T)$,

$$\begin{aligned}\lim_{\Delta\theta\to 0} \frac{1}{\Delta\theta} E\left[\sum_{i=1}^{N(T)} -(\alpha_i - \beta_i)F_i\right] &\geq \lim_{\Delta\theta\to 0} \frac{1}{\Delta\theta} \sum_{i=1}^{N(T)} -c\,(\Delta\theta)^2 \\ &= \lim_{\Delta\theta\to 0} \left[-c\Delta\theta N(T)\right]\end{aligned}$$

and

$$\lim_{\Delta\theta\to 0} \left[-c\Delta\theta\right] E\left[N(T)\right] = 0$$

where, by assumption, $E[N(T)] < \infty$. This completes the proof. ■

An immediate implication of this theorem is that $-B(\theta)$ is an unbiased estimator of $dE[L_T(\theta)]/d\theta$:

$$\left[\frac{dE[L_T(\theta)]}{d\theta}\right]_{est} = -B(\theta) \tag{B.30}$$

This estimator is extremely simple to implement: (B.30) is merely a counter of all busy periods observed in $[0, T]$ in which at least one overflow takes place. Again, no knowledge of the traffic or processing rates is required, nor does (B.30) depend on the nature of the random processes involved.

Using the finite difference approach above, it is also possible to derive an unbiased estimator for $dE[Q_T(\theta)]/d\theta$ (see [15]), but it is considerably more tedious; we will see how to derive the same estimator in the next section by simpler means. Finally, note that (B.28) was derived using $\Delta\theta > 0$; thus, the analysis has to be repeated for $\Delta\theta < 0$ in order to evaluate the left sample derivative, and, although this does not present any conceptual difficulties, it adds to the tediousness of the finite difference analysis we have pursued thus far.

B.4.2 IPA Using Sample Derivatives

In this subsection, we derive explicitly the sample derivatives $L_T'(\theta)$ and $Q_T'(\theta)$ of the loss volume and work, defined in (B.11) and (B.12), respectively. We then show that they provide unbiased estimators of the expected loss volume sensitivity $dE[L_T(\theta)]/d\theta$ and the expected work sensitivity $dE[Q_T(\theta)]/d\theta$.

Since we are concerned with the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$, we have to identify conditions under which they exist. Observe that any endogenous event time (a time point when the buffer becomes full or empty) is generally a function of θ; see also (B.6). Denoting this point by $t(\theta)$, the derivative $t'(\theta)$ exists as long as $t(\theta)$ is not a jump point of the difference process $\{\alpha(t) - \beta(t)\}$. Recall that the times at which the buffer ceases to be full or empty are locally independent of θ, because they correspond to a change-of-sign of the difference sample function $\alpha(t)-\beta(t)$, which does not depend on θ. Excluding the possibility of the simultaneous occurrence of two events, the only situation preventing the existence of the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ involves an interval during which $x(t) = \theta$ and $\alpha(t) - \beta(t) = 0$, as seen in (B.8)); in this case, the one-sided derivatives of $L_T(\theta)$and $Q_T(\theta)$ exist and can be obtained with the approach of the previous section. In order to keep the analysis simple, we focus only on the differentiable case. Therefore, the analysis that follows rests on the following technical conditions:

Assumption 1.
a. W.p.1, $\alpha(t) - \beta(t) \neq 0$.
b. For every $\theta \in \Theta$, w.p.1, no two *events* may occur at the same time.

Remark. We stress the fact that the above conditions for ensuring the existence of the sample derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ are very mild. Part *b* above is satisfied whenever the cdf's (or conditional cdf's) characterizing the intervals between exogenous event occurrences are continuous. For example, in the simple case where $\beta(t) = \beta$ and $\alpha(t)$ can only take two values, 0 and $\alpha > \beta$, suppose that the inflow process switches from α to 0 after $\theta/(\alpha - \beta)$ time units w.p. 1. The buffer then becomes full exactly when an exogenous event occurs, and the loss volume sample function experiences a discontinuity w.p. 1. Such situations can only arise for a small finite subset of Θ (for which one can still calculate either the left or right derivatives) and they are of limited practical consequence.

We next derive the IPA derivatives of $L_T(\theta)$and $Q_T(\theta)$. Recall that $B(\theta) = |\Phi(\theta)|$, i.e., the number of BPs containing at least one overflow period.

Theorem 8 *For every $\theta \in \Theta$,*

$$L_T^{'}(\theta) \;=\; -B(\theta). \tag{B.31}$$

Proof. Recalling that $\mathcal{B}_k = (\xi_k, \eta_k(\theta))$, we have, from (B.11),

$$L_T(\theta) \;=\; \sum_{k\in\Phi(\theta)} \int_{\xi_k}^{\eta_k(\theta)} \gamma(\theta;t)dt, \tag{B.32}$$

which after differentiation yields

$$L_T^{'}(\theta) \;=\; \sum_{k\in\Phi(\theta)} \frac{d}{d\theta} \int_{\xi_k}^{\eta_k(\theta)} \gamma(\theta;t)dt. \tag{B.33}$$

Note that the derivative in (B.33) is taken along a sample path. The set $\Phi(\theta)$, though depending on θ, can be viewed as a constant for the purpose of taking the derivative. The reason is that, by virtue of **Assumption 1***b*, it is locally independent of θ, similarly to the endogenous event times discussed in the first part of Section 4 (i.e., for every fixed θ, w.p.1 there exists $\Delta\theta > 0$, such that, for every $\bar{\theta} \in [\theta - \Delta\theta, \theta + \Delta\theta]$, $\Phi(\bar{\theta}) = \Phi(\theta)$; although this $\Delta\theta$ generally depends on the given sample path, our derivative is taken along a specific sample path, hence (B.33) is justified).

Next, we focus on a particular $\mathcal{B}_k$ with $k \in \Phi(\theta)$ and we shall suppress the index k to simplify the notation. Accordingly, the BP in question is denoted by $\mathcal{B} = (\xi, \eta(\theta))$, and there are $M \geq 1$ overflow periods in $\mathcal{B}$, denoted by $\mathcal{F}_m = [u_m(\theta), v_m]$, $m = 1, \ldots, M$. A typical scenario is depicted in Fig. B.4, where in the first BP we have $M = 2$. The loss volume over $\mathcal{B}$ is given by the function

$$\lambda(\theta) \;=\; \int_{\xi}^{\eta(\theta)} \gamma(\theta; t)dt. \tag{B.34}$$

We next prove that

$$\lambda^{'}(\theta) \;=\; -1, \tag{B.35}$$

from which Eq. (B.31) immediately follows in view of (B.32)-(B.34). From the definition of $\gamma(\theta; t)$ in (B.8), we can rewrite (B.34) as

$$\lambda(\theta) \;=\; \sum_{m=1}^{M} \int_{u_m(\theta)}^{v_m} [\alpha(t) - \beta(t)]dt. \tag{B.36}$$

Since the points $u_m(\theta)$, $m = 1, \ldots, M$, and the jump points of $\alpha(t) - \beta(t)$ constitute *events*, and since w.p. 1 no two events can occur at the same time by **Assumption 1***b*, the function $\alpha(t) - \beta(t)$ must be continuous w.p. 1 at the points $u_m(\theta)$, $m = 1, \ldots, M$. Consequently, by taking derivatives with respect to θ in (B.36) we obtain,

$$\lambda^{'}(\theta) \;=\; -\sum_{m=1}^{M} [\alpha(u_m(\theta)) - \beta(u_m(\theta))]u_m^{'}(\theta). \tag{B.37}$$

Next, consider the individual terms in the above sum (see also Fig. B.4 for an illustration).

1. If $m = 1$, then the buffer is neither full nor empty in the interval $(\xi, u_1(\theta))$. Since the buffer content evolves from $x(\xi) = 0$ to $x(u_1(\theta)) = \theta$, (B.6) implies

   $$\int_{\xi}^{u_1(\theta)} [\alpha(t) - \beta(t)]dt \;=\; \theta,$$

   and, upon taking derivatives with respect to θ,

   $$[\alpha(u_1(\theta)) - \beta(u_1(\theta))]u_1^{'}(\theta) \;=\; 1. \tag{B.38}$$

2. If $m > 1$, then the buffer is neither full nor empty in the interval $(v_{m-1}, u_m(\theta))$. Since $x(v_{m-1}) = x(u_m(\theta)) = \theta$ we obtain, by (B.6),

   $$\int_{v_{m-1}}^{u_m(\theta)} [\alpha(t) - \beta(t)]dt \;=\; 0,$$

and upon differentiating with respect to θ,

$$[\alpha(u_m(\theta)) - \beta(u_m(\theta))]u_m^{'}(\theta) \;=\; 0 \tag{B.39}$$

Finally, Eqs. (B.37), (B.38) and (B.39) imply (B.35), which immediately implies (B.31) and the proof is complete. ■

Note that Theorem 8 is consistent with Theorem 7. However, Theorem 7 includes a direct proof of the unbiasedness of the estimator $-B(\theta)$, whereas the present approach requires a separate proof that the sample derivative $L_T^{'}(\theta) = -B(\theta)$ is in fact unbiased. The unbiasedness of this IPA derivative will be proven later, after we establish the IPA derivative of the work $Q_T(\theta)$ defined in (B.12).

Theorem 9 *For every $\theta \in \Theta$,*

$$Q_T^{'}(\theta) \;=\; \sum_{k\in\Phi(\theta)} [\eta_k(\theta) - u_{k,1}(\theta)]. \tag{B.40}$$

Proof. We focus on a particular BP $\mathcal{B}_k$ with $k \in \Phi(\theta)$, and again suppress the notational dependency on k for the sake of simplicity. Accordingly, consider a BP $\mathcal{B}_k = (\xi, \eta(\theta))$, and denote its overflow periods by $\mathcal{F}_m = [u_m(\theta), v_m]$, $m = 1, \ldots, M$, for some $M \geq 1$ (e.g., $M = 2$ in the first BP of Fig. B.4). Define the function

$$q(\theta) \;=\; \int_\xi^{\eta(\theta)} x(\theta;t)dt. \tag{B.41}$$

It suffices to prove that

$$q'(\theta) \;=\; \eta(\theta) - u_1(\theta) \tag{B.42}$$

since this would immediately imply (B.40). Since $x(\theta;t)$ is continuous in t, taking the derivative with respect to θ in (B.41) and letting $x'(\theta;t)$ denote the partial derivative with respect to θ yields

$$q'(\theta) \;=\; \int_\xi^{\eta(\theta)} x'(\theta;t)dt + x(\theta;\eta(\theta))\eta'(\theta) = \int_\xi^{\eta(\theta)} x'(\theta;t)dt, \tag{B.43}$$

since the BP ends at $\eta(\theta)$, hence $x(\theta;\eta(\theta)) = 0$. To evaluate this partial derivative (which exists at all t except $t = u_m$ and $t = v_m$) we consider all possible cases regarding the location of t in the BP $\mathcal{B}_k = (\xi, \eta(\theta))$ (see Fig. B.4):

1. $t \in (\xi, u_1(\theta))$. In this case, the buffer is neither empty nor full in this interval. It follows, using (B.6), that

$$x(\theta;t) \;=\; \int_\xi^t [\alpha(\tau) - \beta(\tau)]d\tau.$$

Since the right-hand side above is independent of θ, we have $x'(\theta;t) = 0$.

2. $t \in (u_m(\theta), v_m)$, $m = 1 \ldots, M$. Since $(u_m(\theta), v_m)$ is an overflow period, $x(\theta, t) = \theta$ in these intervals, hence $x'(\theta; t) = 1$.

3. $t \in (v_m, u_{m+1}(\theta))$, $m = 1, \ldots, M - 1$. Here, the buffer is neither empty nor full in the interval (v_m, t), while $x(\theta; v_m) = \theta$. It follows, using (B.6), that

$$x(\theta; t) \;=\; \theta \;+\; \int_{v_m}^{t} [\alpha(\tau) - \beta(\tau)]d\tau,$$

   and upon differentiating with respect to θ, we obtain $x'(\theta; t) = 1$.

4. $t \in (v_M, \eta(\theta))$. This case is identical to the previous one, yielding $x'(\theta; t) = 1$.

In summary, $x'(\theta; t) = 0$ for all $t \in (\xi, u_1(\theta))$ (Case 1), and $x'(\theta; t) = 1$ for all $t \in (u_1(\theta), \eta(\theta))$ (Cases 2-4). Therefore, it follows from (B.43) that (B.42) holds, implying (B.40) and completing the proof. ■

In simple terms, the contribution of a BP, $\mathcal{B}_k$, to the sample derivative $Q_T^{'}(\theta)$ in (B.40) is the length of the interval defined by the first point at which the buffer becomes full and the end of the BP. Once again, as in (B.31), observe that the IPA derivative $Q_T^{'}(\theta)$ is nonparametric, since it requires only the recording of times at which the buffer becomes full (i.e., $u_{k,1}(\theta)$) and empty (i.e., $\eta_k(\theta)$) for any $\mathcal{B}_k$ with $k \in \Phi(\theta)$. We also remark that the same IPA derivative can be obtained through the finite difference analysis of the previous section (see [15]), but with considerably more effort.

IPA Unbiasedness

We next prove the unbiasedness of the IPA derivatives $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ obtained above. Although we have already shown in (B.28) that $-B(\theta)$ is an unbiased estimate of $dE[L_T(\theta)]/d\theta$, we supply an alternative and greatly simplified proof based on the direct derivation of the IPA estimator in this section and on some of the results of the finite-difference analysis in Section 4.1. By a similar technique, we also supply a proof of the unbiasedness of the IPA estimator $Q_T^{'}(\theta)$ in (B.40). These proofs, jointly with the sample-derivative technique for obtaining the estimators, suggest the possibility of extensive generalizations to the functional forms of $\alpha(t)$ and $\beta(t)$ (beyond piecewise constant), to be explored in a forthcoming paper (also, see [78], [79]).

In general, the unbiasedness of an IPA derivative $\mathcal{L}'(\theta)$ has been shown to be ensured by the following two conditions (see [71], Lemma A2, p.70):

Condition 1. For every $\theta \in \Theta$, the sample derivative $\mathcal{L}'(\theta)$ exists w.p.1.

Condition 2. W.p.1, the random function $\mathcal{L}(\theta)$ is Lipschitz continuous throughout Θ, and the (generally random) Lipschitz constant has a finite first moment.

Consequently, establishing the unbiasedness of $L_T^{'}(\theta)$ and $Q_T^{'}(\theta)$ as estimators of $dE[L_T(\theta)]/d\theta$ and $dE[Q_T(\theta)]/d\theta$, respectively, reduces to verifying the Lipschitz continuity of $L_T(\theta)$ and $Q_T(\theta)$ with appropriate Lipschitz constants. Recall that $N(T)$ is the random number of all exogenous events in $[0, T]$ and that we have assumed $E[N(T)] < \infty$.

Theorem 10 *Under **Assumption 1**,*

1. *If $E[N(T)] < \infty$, then the IPA derivative $L_T^{'}(\theta)$ is an unbiased estimator of $dE[L_T(\theta)]/d\theta$.*
2. *The IPA derivative $Q_T^{'}(\theta)$ is an unbiased estimator of $dE[Q_T(\theta)]/d\theta$.*

Proof. Under **Assumption 1**, **Condition 1** holds for $L_T(\theta)$ and $Q_T(\theta)$. Therefore, it only remains to establish **Condition 2**.

First, consider $L_T(\theta)$. Recalling (B.14) and (B.15), we can write

$$\Delta L_T(\theta) = \sum_{i=1}^{N(T)} \Delta L_i(\theta)$$

by partitioning $[0, T]$ into intervals $[A_{i-1}, A_i)$ defined by successive exogenous events. Then, by Lemma 5, $-\Delta\theta \leq \Delta L_i \leq 0$, so that

$$|\Delta L_T(\theta)| \leq N(T) \, |\Delta\theta| \, ,$$

i.e., $L_T(\theta)$ is Lipschitz continuous with constant $N(T)$. Since $E[N(T)] < \infty$, this establishes unbiasedness.

Consider next the sample function $Q_T(\theta)$, defined by (B.12) and fix θ and $\Delta\theta > 0$. By Lemma 4, $0 \leq \Delta x_i \leq \Delta\theta$, hence the difference $\Delta x(\theta, \Delta\theta; t) := x(\theta + \Delta\theta; t) - x(\theta; t)$ satisfies the inequalities

$$0 \ \leq \ \Delta x(\theta, \Delta\theta; t) \ \leq \ \Delta\theta.$$

Consequently, in view of (B.12),

$$|\Delta Q_T(\theta)| = \left| \int_0^T \Delta x(\theta, \Delta\theta; t) dt \right| \leq T \, |\Delta\theta| \, ,$$

that is, $Q_T(\theta)$ is Lipschitz continuous with constant T. This completes the proof. ■

Remark. For the more commonly used performance metrics $\frac{1}{T} E\left[L_T(\theta)\right]$ (the Expected Loss Rate over $[0, T]$) and $\frac{1}{T} E\left[Q_T(\theta)\right]$ (the Expected Buffer Content over $[0, T]$), the Lipschitz constants in Theorem 10 become $N(T)/T$ and 1, respectively. As $T \to \infty$, the former quantity typically converges to the exogenous event rate.

B.5 Optimal Buffer Control Using SFM-Based IPA Estimators

As suggested in Section 2 and illustrated in Fig. B.2, the solution to an optimization problem defined for an actual network node (i.e., a node that operates as a queueing system) may be accurately approximated by the solution to the same problem based on a SFM of the node. However, this may not be always the case. On the other hand, the simple form of the IPA estimators of the Expected Loss Rate and Expected Buffer Content obtained through (B.31) and (B.40) allows us to use data from the *actual* (real-world) system in order to estimate sensitivities that, in turn, may be used to solve an optimization problem of interest. In other words, the *form* of the IPA estimators is obtained by analyzing the system as a SFM, but the associated *values* are based on real data. In particular, an algorithm for implementing the estimators (B.31) and (B.40) is given below:

IPA Estimation Algorithm

- Initialize a counter $\mathcal{C} := 0$ and a cumulative timer $\mathcal{T} := 0$.
- Initialize $\tau := 0$.
- If an overflow event is observed at time t and $\tau = 0$:
    - Set $\tau := t$
- If a busy period ends at time t and $\tau > 0$:
    - Set $\mathcal{C} := \mathcal{C} - 1$ and $\mathcal{T} := \mathcal{T} + (t - \tau)$
    - Reset $\tau := 0$.
- If $t = T$, and $\tau > 0$:
    - Set $\mathcal{C} := \mathcal{C} - 1$ and $\mathcal{T} := \mathcal{T} + (t - \tau)$.

The final values of $\mathcal{C}$ and $\mathcal{T}$ provide the IPA derivatives $L_T'(\theta)$ and $Q_T'(\theta)$ respectively. We remark that the "overflow" and "end of BP" events are readily observable during actual network operation. In addition, we point out once again that these estimates are independent of all underlying stochastic features, including traffic and processing rates. Finally, the algorithm is easily modified to apply to any interval $[T_1, T_2]$.

Let us now return to the buffer control problem presented in Section 2, where the objective was to determine a threshold C that minimizes a cost function of the form

$$J_T(C) \;=\; \bar{Q}_T(C) \;+ R \cdot \bar{L}_T(C)$$

trading off the expected loss rate with a rejection penalty R for the expected queue length. If a SFM is used instead, then the cost function of interest becomes

$$J(\theta) = \frac{1}{T} E[Q_T(\theta)] + \frac{R}{T} E[L_T(\theta)]$$

and the optimal threshold parameter, θ^*, may be determined through a standard stochastic approximation algorithm based on (B.4). The gradient estimator $H_n(\theta, \omega_n^{SFM})$ is the IPA estimator of $dJ/d\theta$ based on (B.31) and (B.40):

$$H_n(\theta, \omega_n^{SFM}) = \frac{1}{T} \sum_{k \in \Phi(\theta)} [\eta_k(\theta) - u_{k,1}(\theta)] - \frac{R}{T} B(\theta) \tag{B.44}$$

evaluated over a simulated sample path ω_n^{SFM} of length T, following which a control update is performed through (B.4) based on the value of $H_n(\theta, \omega_n^{SFM})$.

The interesting observation here is that the same estimator may be used in (B.5) as follows: If a packet arrives and is rejected, the time this occurs is recorded as τ in the algorithm above. At the end of the current busy period, the counter $\mathcal{C}$ and timer $\mathcal{T}$ are updated. Thus, the exact same expression as in the right-hand side of (B.44) can be used to update the threshold:

$$C_{n+1} = C_n - \nu_n H_n(C_n, \omega_n^{DES}), \qquad n = 0, 1, \ldots \tag{B.45}$$

Note that, after a control update, the state must be reset to 0, in accordance with our convention that all performance metrics are defined over an interval $[0, T]$ with an initially empty buffer. In the case of off-line control, this simply amounts to simulating the system after resetting its state to 0. In the more interesting case of on-line control, we proceed as follows. Suppose that the nth iteration ends at time τ_n and the state is $x(C_n; \tau_n)$ (in general, $x(C_n; \tau_n) > 0$). At this point, the threshold is updated and its new value is C_{n+1}. Let $\tau_n^0 > \tau_n$ be the next time that the buffer is empty, i.e., $x(C_{n+1}; \tau_n^0) = 0$. At this point, the $(n+1)$th iteration starts and the next gradient estimate is obtained over the interval $[\tau_n^0, \tau_n^0 + T]$, so that $\tau_{n+1} = \tau_n^0 + T$ and the process repeats. The implication is that over the interval $[\tau_n, \tau_n^0]$ no estimation is carried out while the controller waits for the system to be reset to its proper initial state; therefore, sample path information available over $[\tau_n, \tau_n^0]$ is effectively wasted as far as gradient estimation is concerned.

Figure B.6 depicts examples of the application of this scheme to a single-node SFM under six different parameter settings (scenarios), summarized in Table 1. As in Fig. B.2, 'DES' denotes curves obtained by estimating $J_T(C)$ over different (discrete) values of C, 'SFM' denotes curves obtained by estimating $J(\theta)$ over different values of θ, and 'Opt.Algo.' represents the optimization process (B.45), where we maintain real-valued thresholds throughout. The first three scenarios correspond to a high traffic intensity ρ compared to the remaining three. For each example, C^* is the optimal threshold obtained through exhaustive simulation. In all simulations, an ON-OFF traffic source is used with the number of arrivals in each ON period geometrically distributed with parameter p and arrival rate α; the OFF period is exponentially distributed with parameter μ; and the service rate is fixed at β. Thus, the traffic intensity of the system ρ is $\alpha(\frac{1}{\alpha p})/\beta(\frac{1}{\alpha p} + \frac{1}{\mu})$, where $\frac{1}{\alpha p}$ is the average length of an ON period and $\frac{1}{\mu}$ is the average length of an OFF period. The rejection cost is $R = 50$. For simplicity, ν_n in (B.45) is taken to be a constant $\nu_n = 5$. Finally, in all cases $T = 100,000$. As seen in Fig. B.6, the threshold value obtained through (B.45) using the SFM-based gradient estimator in (B.44) either recovers C^* or is close to it with a cost value extremely close to $J_T(C^*)$; since in some cases the cost function is nearly constant in the neighborhood of the optimum, it is difficult to determine the actual optimal threshold, but it is also practically unimportant since the cost is essentially the same. We have also implemented (B.45) with $H_n(C_n, \omega_n^{DES})$ estimated over shorter interval lengths $T = 10,000$ and $T = 5,000$, with virtually identical results. Looking at Fig. B.6, it is worth observing that determining θ^* as an approximation to C^* through off-line analysis of the SFM would also yield good approximations, further supporting the premise of this paper that SFMs pro-

Scenario	ρ	α	p	μ	β	C^*
1	0.99	1	0.1	0.1	0.505	7
2	0.99	1	0.05	0.05	0.505	7
3	0.99	2	0.05	0.1	1.01	15
4	0.71	1	0.1	0.1	0.7	13
5	0.71	1	0.05	0.05	0.7	11
6	0.71	2	0.05	0.1	1.4	22

Table B.1: Parameter settings for six examples

vide an attractive modeling framework for control and optimization (not just performance analysis) of complex networks.

B.6 Conclusions and Future Work

Stochastic Fluid Models (SFM) can adequately describe the dynamics of high-speed communication networks, where they may be used to approximate discrete event models or constitute primary models in their own right. When control and optimization are of primary importance (rather than performance analysis), a SFM may be used as a means for accurately determining an optimal parameter setting, even though the corresponding performance evaluated through the SFM may not be particularly accurate. With this premise in mind, we have considered single-node SFMs from the standpoint of IPA derivative estimation. In particular, we have developed IPA estimators for the loss volume and work as functions of the buffer size, and shown them to be unbiased and nonparametric. The simplicity of the estimators and their nonparametric property suggest their application to on-line network management. Indeed, for a class of buffer control problems, we have shown how to use an optimization scheme (and illustrated it through numerical examples) for a discrete event model (viewed as a real, queueing-based single-node system) using the IPA gradient obtained from its SFM counterpart. Interestingly, there is no IPA derivative for the discrete event model, since its associated control parameter is discrete.

For the loss volume performance function, the IPA derivative has been developed by two separate techniques: finite difference analysis, and a sample derivative analysis. The former method is more elaborate, but sheds light on the structure of the derivative estimator. The second method is more direct and elegant, but its unbiasedness proof requires some results obtained by the analysis of the former method. The sample-derivative method was also applied to the IPA estimator of the buffer workload performance function.

The sample derivative analysis holds the promise of considerable extensions to multiple SFMs as models of actual networks and to multiple flow classes that can be used for differentiating traffic classes with different Quality-of-Service (QoS) requirements. Ongoing research has already led to very encouraging results, reported in [15], involving IPA estimators and associated optimization for flow control purposes in multi-node models. Finally, for the purpose of session-by-session admission control, preliminary work suggests that one

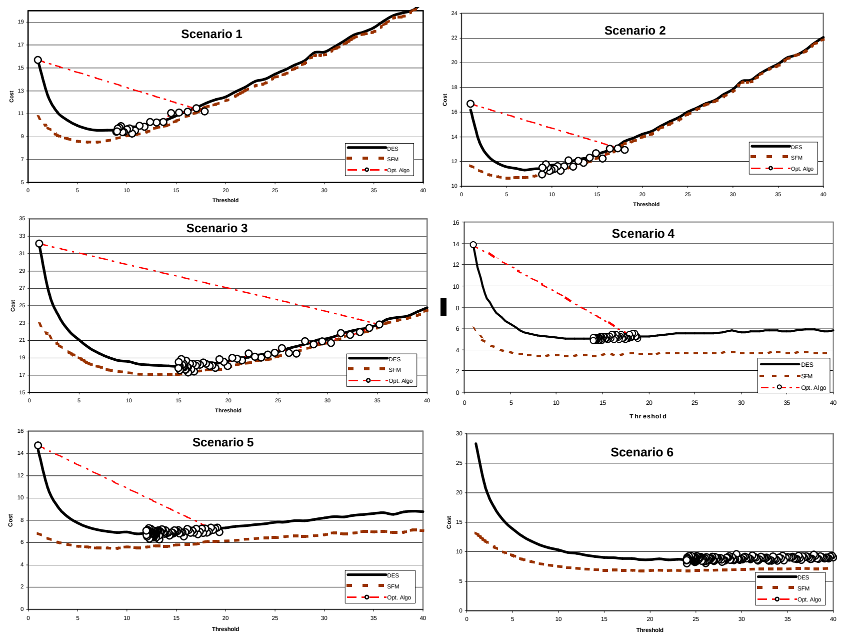


Figure B.6: Optimal threshold determination in an actual system using SFM-based gradient estimators - Scenarios 1-6

can use sensitivity information with respect to inflow rates (which can be obtained through an approach similar to the one presented in this paper) and contribute to the development of effective algorithms, yet to be explored.

Appendix

Proof of Lemma 4: Looking at any segment of the sample path over an interval $[A_i, A_{i+1})$, there are two possibilities: either $\alpha_i - \beta_i < 0$ or $\alpha_i - \beta_i \geq 0$. First, suppose that $\alpha_i - \beta_i < 0$ and consider the event which occurs at time A_{i+1}. There are three cases to analyze:

Case 1.1: $y_{i+1}(\theta) \geq 0$. In this case, as seen in Fig. B.7, we have:

$$\Delta y_{i+1} = \Delta x_{i+1} = \Delta x_i \tag{B.46}$$

Case 1.2: $y_{i+1}(\theta) < 0$ and $y_{i+1}(\theta) + \Delta y_{i+1} \leq 0$. In this case, as seen in Fig. B.8(a), the kth BP ends and it is followed by an EP of length I_i, which in turn ends at time A_{i+1}. Clearly,

$$\Delta x_{i+1} = 0 \tag{B.47}$$

Case 1.3: $y_{i+1}(\theta) < 0$ and $y_{i+1}(\theta) + \Delta y_{i+1} > 0$. This represents a situation where an EP of length I_i is eliminated in the perturbed path, i.e., $I_i < \Delta x_i/(\beta_i - \alpha_i)$. As seen in Fig.

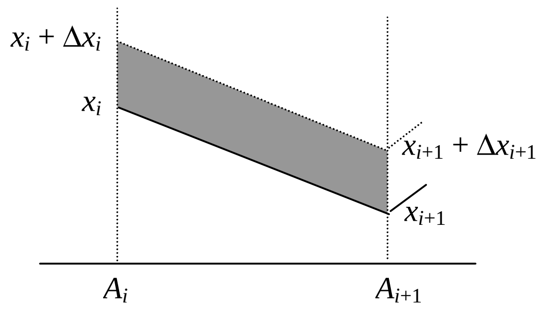


Figure B.7: Case 1.1: $y_{i+1}(\theta) \geq 0$

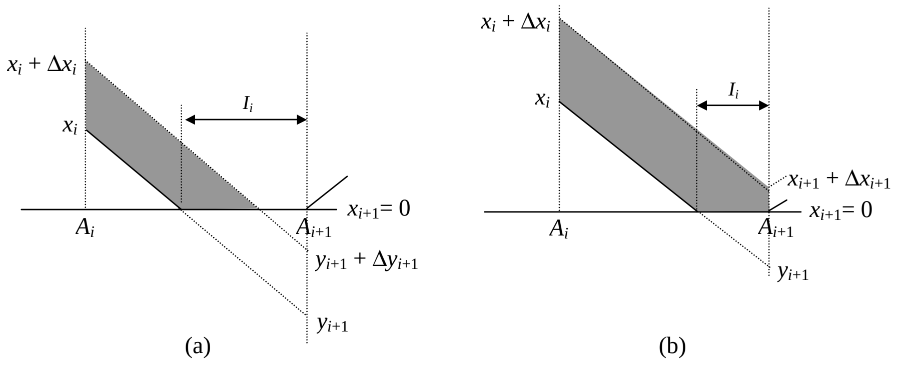


Figure B.8: (a) Case 1.2: $y_{i+1}(\theta) < 0$ and $y_{i+1}(\theta) + \Delta y_{i+1} \leq 0$. (b) Case 1.3: $y_{i+1}(\theta) < 0$ and $y_{i+1}(\theta) + \Delta y_{i+1} > 0$

B.8(b), the buffer content perturbation becomes

$$\Delta x_{i+1} = \Delta x_i - (\beta_i - \alpha_i) I_i, \tag{B.48}$$

Next, let us assume that $\alpha_i - \beta_i \geq 0$. We then have three cases as follows:

Case 2.1: $y_{i+1}(\theta) \leq \theta$ and $y_{i+1}(\theta) + \Delta y_{i+1} \leq \theta$. It is easy to see (Fig. B.9(a)) that this is identical to **Case 1.1** yielding (B.46).

Case 2.2: $y_{i+1}(\theta) \leq \theta$ and $y_{i+1}(\theta) + \Delta y_{i+1} > \theta$. The perturbed buffer content cannot exceed $\theta + \Delta\theta$, since $\Delta y_{i+1} = \Delta x_i \leq \Delta\theta$ from (B.21); therefore, $y_{i+1} + \Delta y_i \leq \theta + \Delta\theta$ and the situation is identical to that of Fig. B.9(a), again yielding (B.46). **Case 2.3:** $y_{i+1}(\theta) > \theta$. As seen in Fig. B.9(b),

$$\Delta x_{i+1} = \Delta\theta$$

as in **Case II** where perturbation generation was considered. Once again, however, it is possible that $\Delta\theta > (\alpha_i - \beta_i) F_i + \Delta x_i$, so that we write, similarly to **Case II**,

$$\Delta x_{i+1} = \Delta\theta - \left[\Delta\theta - \Delta x_i - (\alpha_i - \beta_i) F_i\right]^+ . \tag{B.49}$$

We may now establish (B.23) by combining (B.46), (B.47), (B.48), and (B.49) and by observing that (*i*) In (B.48), $I_i < \Delta x_i / (\beta_i - \alpha_i)$ with $\beta_i - \alpha_i > 0$, therefore $0 < \Delta x_{i+1} < \Delta x_i$, and (*ii*) In (B.49), $0 \leq \Delta x_{i+1} \leq \Delta\theta$.

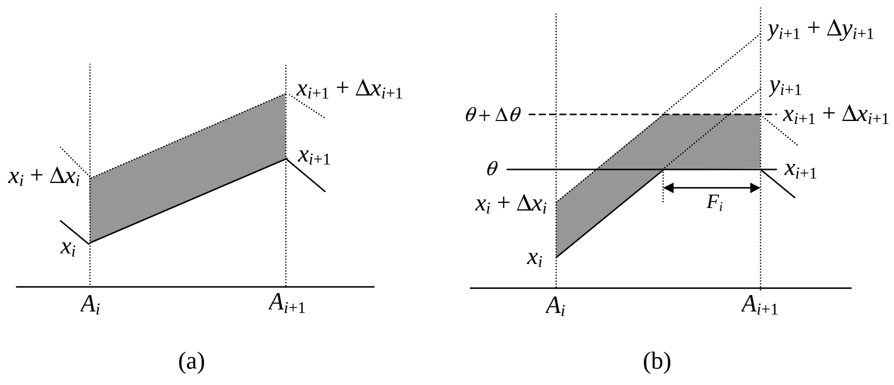


Figure B.9: (a) Cases 2.1-2.2: $y_{i+1}(\theta) \leq \theta$. (b) Case 2.3: $y_{i+1}(\theta) > \theta$

Next, by combining (B.46), (B.47), (B.48) we obtain the first part of (B.24), observing that $I_i = 0$ in **Case 1.1**. To obtain the second part, we combine (B.46) and (B.49), observing that when $F_i = 0$ in (B.49), we get $\Delta x_{i+1} = \Delta x_i$, since $\Delta\theta - \Delta x_i \geq 0$ from (B.23), which reduces to (B.46) corresponding to **Cases 2.1-2.2**. ∎

Proof of Lemma 5: Proceeding as in the proof of Lemma 4, we first consider the case $\alpha_i - \beta_i < 0$ and get:

Case 1.1: $y_{i+1}(\theta) \geq 0$. In this case, as seen in Fig. B.7, we have:

$$\Delta L_{i+1} = 0 \tag{B.50}$$

Case 1.2: $y_{i+1}(\theta) < 0$ and $y_{i+1}(\theta) + \Delta y_{i+1} \leq 0$. Clearly, as seen in Fig. B.8(a),

$$\Delta L_{i+1} = 0 \tag{B.51}$$

Case 1.3: $y_{i+1}(\theta) < 0$ and $y_{i+1}(\theta) + \Delta y_{i+1} > 0$. This represents a situation where an EP of length I_i is eliminated in the perturbed path, i.e., $I_i < \Delta x_i/(\beta_i - \alpha_i)$. As seen in Fig. B.8(b), no loss is involved in either path:

$$\Delta L_{i+1} = 0. \tag{B.52}$$

Next, let us assume that $\alpha_i - \beta_i \geq 0$ and we have:

Case 2.1: $y_{i+1}(\theta) \leq \theta$ and $y_{i+1}(\theta) + \Delta y_{i+1} \leq \theta$. It is easy to see (Fig. B.9(a)) that this is identical to **Case 1.1** yielding (B.50).

Case 2.2: $y_{i+1}(\theta) \leq \theta$ and $y_{i+1}(\theta) + \Delta y_{i+1} > \theta$. As argued in the proof of Lemma 4, the situation is identical to that of Fig. B.9(a), again yielding (B.50).

Case 2.3: $y_{i+1}(\theta) > \theta$. If $\Delta\theta > (\alpha_i - \beta_i)F_i + \Delta x_i$, then $\Delta L_{i+1} = 0 - (y_{i+1} - \theta) = -(\alpha_i - \beta_i)F_i$. Otherwise, $L_{i+1}(\theta + \Delta\theta) = y_{i+1} + \Delta x_i - \theta - \Delta\theta$, and we get $\Delta L_{i+1} = \Delta x_i - \Delta\theta$. Thus,

$$\Delta L_{i+1} = (\Delta x_i - \Delta\theta) + [\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i]^+ . \tag{B.53}$$

We may now combine (B.50), (B.51), (B.52), (B.53). Observe that in (B.53) $\Delta L_{i+1} \geq -\Delta\theta$, since we have already established that $\Delta x_i \geq 0$ in Lemma 4. Moreover, $\Delta L_{i+1} = -(\alpha_i - \beta_i)F_i < 0$ if $\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i > 0$, and $\Delta L_{i+1} = \Delta x_i - \Delta\theta$ if $\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i \leq 0$, where $\Delta x_i - \Delta\theta \leq 0$ from Lemma 4. This yields (B.26). ∎

Proof of Lemma 6: Proceeding as in the proof of Lemma 5, we get $\Delta L_{i+1} = 0$ in (B.50), (B.51), (B.52), i.e., in all cases except **Case2.3** where (B.53) applies:

$$\Delta L_{i+1} = (\Delta x_i - \Delta\theta) + [\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i]^+$$

Suppose the first overflow interval in the BP ends at A_r. Under the assumption $\Delta\theta - \Delta x_i - (\alpha_i - \beta_i)F_i \leq 0$ for all $i = j, \ldots, m$, it follows from (B.17)-(B.18) that $\Delta x_r = \Delta\theta$ and $\Delta L_r = -\Delta\theta$. Moreover, from Lemma 4, (B.24) gives $\Delta x_i = \Delta\theta$ for all $i = r+1, \ldots, m$. Therefore, (B.53) gives $\Delta L_{i+1} = 0$ after every subsequent overflow interval and we get $\Lambda_k(\Delta\theta) = \Delta L_r = -\Delta\theta$. ∎

Appendix C

Clustering Methods for Multi-Resolution Simulation Modeling

C.G. Cassandras[1], C.G. Panayiotou
Dept of Manufacturing Eng., Boston University, Boston, MA 02215
G. Diehl
Network Dynamics, Inc., 10 Speen Street, Framingham, MA 01701
W-B. Gong, Z. Liu, and C. Zou
Dept of ECE, University of Massachusetts, Amherst, MA 01003

Abstract

Simulation modeling of complex systems is receiving increasing research attention over the past years. In this paper, we discuss the basic concepts involved in multi-resolution simulation modeling of complex stochastic systems. We argue that, in many cases, using the average over all available high-resolution simulation results as the input to subsequent low-resolution modules is inappropriate and may lead to erroneous final results. Instead high-resolution output data should be classified into groups that match underlying patterns or features of the system behavior before sending group averages to the low-resolution modules. We propose high-dimensional data clustering as a key interfacing component between simulation modules with different resolutions and use unsupervised learning schemes to recover the patterns for the high-resolution simulation results. We give some examples to demonstrate our proposed scheme.

Key words: Hierarchical simulation, multi-resolution simulation, clustering.

[1]This work was supported by AFRL under Contracts F30602-99-C-0056 and F30602-99-C-0057, by AFOSR under contract F49620-98-1-0387 and by ERPI/DoD under contract WO8333-03.

C.1 Introduction

In modeling complex systems it is impossible to mimic every detail through simulation. The common approach is to divide the whole system hierarchically into simpler modules, each with different simulation resolution. In this context, the output of a module becomes an input parameter to another, as illustrated in Figure C.1. The decomposed modules can be high-resolution or low-resolution models. High-resolution, e.g. the usual discrete-event simulation models, take detailed account of all possible events, but are generally time consuming. Low-resolution (or coarser) modules, perform aggregate evaluation of the module's functionality (i.e., determine what would happen "on the average"). Such modules are less time consuming and can be any of the following components: differential equations (used for example in combat [76] and semiconductor simulations [54]), standard discrete-event simulation, and fluid simulation [86]. Furthermore, the decomposed modules can also be an optimization or decision support tool such as the one described by Griggs et. al[37].

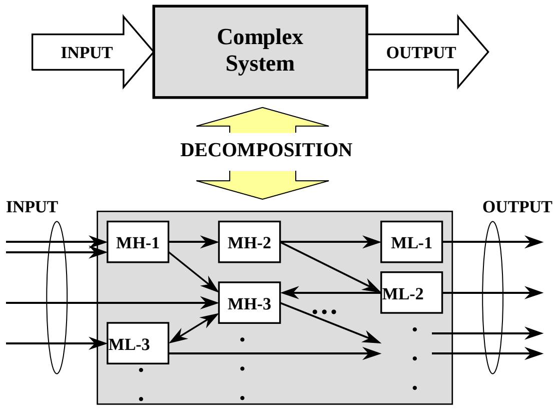


Figure C.1: Decomposition of complex systems

In a hierarchical setting, the lower level simulator (typically a high-resolution model) generates output data which are then taken as input for the higher level simulator (typically a low-resolution model). Hierarchical simulation is a common practice, but the design of hierarchy is always *ad hoc*. A popular practice is to use the mean values of variables from the lower level output as the input to the higher level. This implies that significant statistical information (i.e., statistical fidelity) is lost in this process, resulting in potentially completely inaccurate results. Especially when the ultimate output of the simulation process is of the form 0 or 1 (e.g., "lose" or "win" a combat), such errors can provide the exact opposite of the real output.

A systematic design and analysis framework is definitely needed. In this paper, we present some fundamental components of such a framework. Our effort has been directed at developing an *interface* between two simulation levels to preserve statistical fidelity to the maximum extent that available computing power allows. Our research focus is to use high-dimensional clustering techniques to group the high-resolution sample paths into meaningful clusters and pass on to lower resolution module(s). In the following we explain in more detail this approach.

Quite often, the system being simulated is such that the high-resolution model produces so widely divergent outputs that it does not make sense to summarize such output through a single average over the entire sample space. For example when simulating a combat, it does not make any sense to take the average of the output under different weather conditions. In such cases, we must subdivide the sample space into segments, and get the high-resolution model to produce an appropriate input to the low-resolution model for each such segment. Essentially, the low-resolution model will be broken down into a number of distinct components, one for each segment of the sample space. To carry out such a segmentation, the high-resolution paths first need to be grouped by their common features. These features then determine and feed the corresponding low-resolution model. The practice of classifying objects according to perceived similarities is the basis for much of science and engineering, since organizing data into sensible groupings is one of the most fundamental modes of understanding and learning. Clustering methods have been widely applied in pattern recognition, image processing, and artificial intelligence. In this paper, we deal with clustering methods for the preservation of statistics in hierarchical simulation.

In the following we describe the idea of using the Adaptive Resonance Theory (ART) neural network for this purpose. ART neural networks were developed by Carpenter and Grossberg [10] to understand the clustering function of the human visual system. They are based on a competitive learning scheme and are designed to deal with the stability/plasticity dilemma in clustering and general learning. It is clear that too much stability would lead to a "stubborn" mind, while too much plasticity would lead to unstable learning. ART neural networks successfully resolve this dilemma by matching the input pattern with the prototypes. If the matching is not adequate, a new prototype is created. In this way, previously learned memories are not eroded by new learning. In addition, the ART neural network implements a feedback mechanism during learning to enhance stability.

Our experiments of using ART neural networks with combat simulation paths have been quite successful [14]. We believe further improvement with the ART structure can lead to a fundamental breakthrough in large data clustering, which is needed in complex systems modeling. ART performs the clustering function based on the "angle" between the vectors that describe the various input patterns. In some cases, as it will be discussed in the sequel, this may pose a limitation of ART since the magnitude of an input pattern may contain significant information which is ignored. To alleviate this shortfall we develop a heuristic that allows the magnitude of the input pattern to play a role in the clustering function. Furthermore, we are developing a generic numerical clustering tool, based on the ART neural network, that can be used for many important problems in intelligent data analysis.

In general, the description of a typical sample path generated by a discrete-event system requires a large amount of data since such sample paths are typically quite long. This implies that the dimension of each input pattern will also be large. However for high dimensional data most of the clustering algorithms (including ART) will involve huge computational effort; thus they are not practical for simulation modeling purposes. For this reason we develop a new clustering approach where we try to take advantage of the statistical structure behind a typical sample path. For high dimensional complex systems we try to use a Hidden

Markov Model (HMM), which has be successfully used in speech recognition and other areas[69], to characterize each observed sample path. In our approach, we use an HMM to describe an arbitrary sample path and thus we cluster together all sample paths whose corresponding HMM have a high *similarity measure* (to be defined in Section C.6). The advantage of this approach is that the amount of data required to describe an HMM is generally much smaller than the amount of data required to explicitly describe an observed sample path and as a result the HMM approach is more efficient.

The remaining of this paper is organized as follows. Section C.2 discusses some of the issues that arise when interfacing between modules with different resolution while Section C.3 presents an example where statistical fidelity is lost as a result of poor interfacing between high- and low-resolution models, i.e., passing only averages from high- to low-resolution models. Section C.4 briefly presents the ART clustering algorithm and Section C.5 describes an ART based tool that we have developed and its application on a complex manufacturing system. Section C.6 presents a new approach for clustering sample paths based on HMMs and finally Section C.7 summarizes our work.

C.2 Interface between high- and low-resolution models

As mentioned above, the key issue in hierarchical simulation is the design of the interface between the hierarchies. In a typical hierarchical simulation model, the lower lever consist of a high-resolution model, such as the discrete event simulator, that generates several sample paths given some input parameters $\mathbf{u}$. The output of such simulation models is then used as input to the higher level model (typically a low-resolution model). The question that arises, the focal point of this paper, is how much and what information we need to pass from the high-resolution to the low-resolution model such that statistical fidelity is preserved.

Note that each sample path generated by the high-resolution model is also a function of some randomness ω (a random number sequence generated through some random seed). Thus, any function evaluated over an observed sample path (e.g., $h(\mathbf{u}, \omega)$) is also a random variable. Typically, we are not interested in the value of $h(\mathbf{u}, \omega)$ obtained from a single sample path but rather the expectation $E\{h(\mathbf{u}, \omega)\}$. Based on this, in hierarchical simulation it is customary to use $E\{h(\mathbf{u}, \omega)\}$ as an input parameter to the higher level model as seen in Figure C.2. This is often highly unsatisfactory, since the mean often obscures important features of the high-resolution output. Said in another way, we are seeking $E\{L(h(\mathbf{u}, \omega))\}$, where $L(\cdot)$ is a function corresponding to the low-resolution model, but what we end up evaluating by passing a single average is $L(E\{h(\mathbf{u}, \omega)\})$; however, in general $E\{L(h(\mathbf{u}, \omega))\} \neq L(E\{h(\mathbf{u}, \omega)\})$.

To solve this problem we propose the use of clustering to identify groups of sample paths that have some "common features", and therefore, when averaged together do not cause the loss of too much information. This approach in shown in Figure C.3. From the N observed sample paths we identify $m < N$ groups that share some common features and determine m input parameters $\mathbf{a}_1, \cdots, \mathbf{a}_m$ where $\mathbf{a}_i = E\{h^i(\mathbf{u}, \omega))\}$ and $h^i(\cdot)$ identifies all sample paths in cluster i. Subsequently, each parameter $\mathbf{a}_i$ is used as an input to a lower

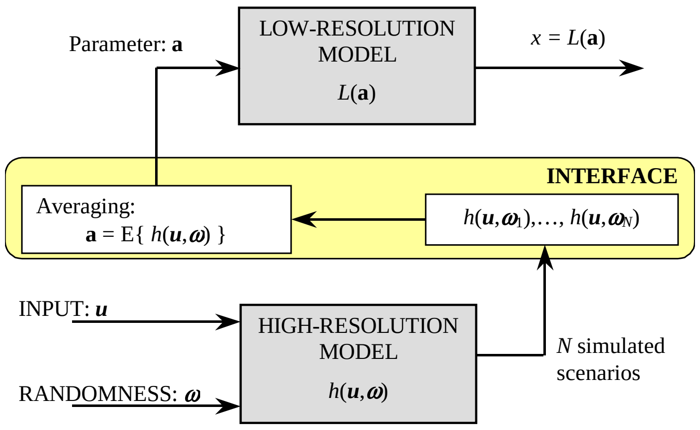


Figure C.2: Hierarchical model interface: passing a simple average to the lower resolution model

resolution model and finally we obtain $E\{L(\mathbf{a}_i)\}$ over m low-resolution components, which we claim is a better estimate of the overall system output than the one obtained using a single average.

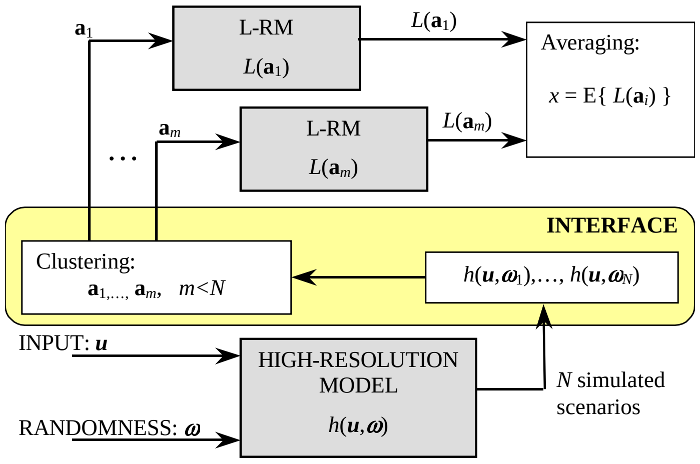


Figure C.3: Hierarchical model interface: passing several averages to the lower resolution model, one for each cluster

One may pose the following question: Since the desired output is of the form $E\{L(h(\mathbf{u}, \omega))\}$ why bother with clustering at all when we can evaluate $L(h(\mathbf{u}, \omega))$ for *all* N obtained samples and then perform the required expectation, especially since the low-resolution models are generally easy to evaluate? The answer to this question lies in the derivation of the low-resolution model. Typically, $L(\mathbf{a})$ assumes that $\mathbf{a}$ is an expectation and therefore it would be meaningless to use some quantity obtained from a single sample path.

C.3 Losing statistical fidelity: an example

In this section we present a simple example where the loss of statistical fidelity can result in poor use of resources with possible catastrophic consequences. For this example, we assume that we are interested in planning a mission that consists of several operations $\mathbf{O}_1, \cdots, \mathbf{O}_N$ as shown in Figure C.4. Due to the dependence between operations (e.g., $\mathbf{O}_4$ cannot start until $\mathbf{O}_2$ and $\mathbf{O}_3$ are completed), it is required to know the time requirements of each operation. So it is natural to ask the following question: *How much time should be allocated for each operation so that the probability of not completing an operation within the allocated time (threshold/deadline) is less than P_0?*

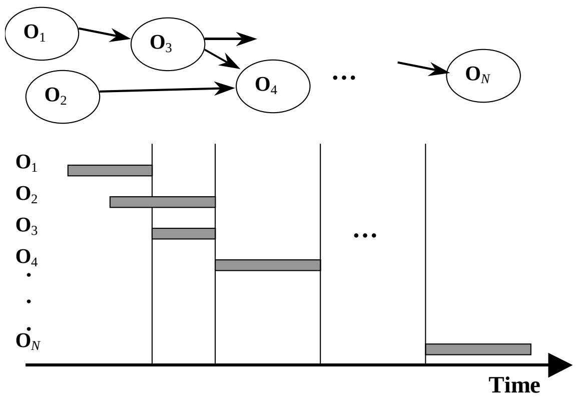


Figure C.4: Operation Scheduling

The methodology for attacking this problem follows the hierarchical structure described next. For every individual operation there exists a high-resolution model that simulates the operation under different scenaria and returns the average m and standard deviation σ of the time that it takes to complete it. Subsequently, the planners use a low-resolution model to determine the time to allocate to the operation so that the probability of not meeting the deadline is less than P_0. One such low-resolution model is the Normal distribution, and the probability of not meeting the scheduled deadline is given by the corresponding cumulative distribution function. Hence, the problem is to find the threshold time T such that

$$F(T) = 1 - \int_{-\infty}^{T} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-m}{\sigma}\right)^2} dx \leq P_0 \tag{C.1}$$

A typical operation in a mission involves the Aircraft Maintenance and Refueling System (ARMS) shown in Figure C.5. In this system there are Q classes $\{C_1, \cdots, C_Q\}$ of aircraft that can be refueled/maintained in one of B bases $\{B_1, \cdots, B_B\}$. Before an aircraft receives any service, it needs to travel to the corresponding base for a time τ which depends on the initial location of the aircraft with respect to the base. Once an aircraft (say aircraft a) arrives at the base, it is assigned one of the M priorities $\{P_1, \cdots, P_M\}$ and it is placed in the corresponding queue. If a token is available at the token queue, it is assigned to a which then proceeds to the next FIFO queue. Once all preceding aircraft are serviced, a

enters the server where it stays for a random period of time with mean $\frac{1}{\mu_{bc}}$ which depends on the base b and the aircraft class c. For a given mission, it is required to service A aircraft before the next operation can start. Therefore, one can use a simulation model to determine the time it takes to process the A aircraft. Note that this processing time depends on several parameters: (a) The number of bases and the routing algorithm that determines what aircraft goes to what base. (b) The initial location of each aircraft and (c) The service times of each class of aircraft at each base. These parameters are considered as the *initial conditions* of each simulation (high-resolution model) and the output is going to be used as an input to the low-resolution model (i.e., the Normal distribution).

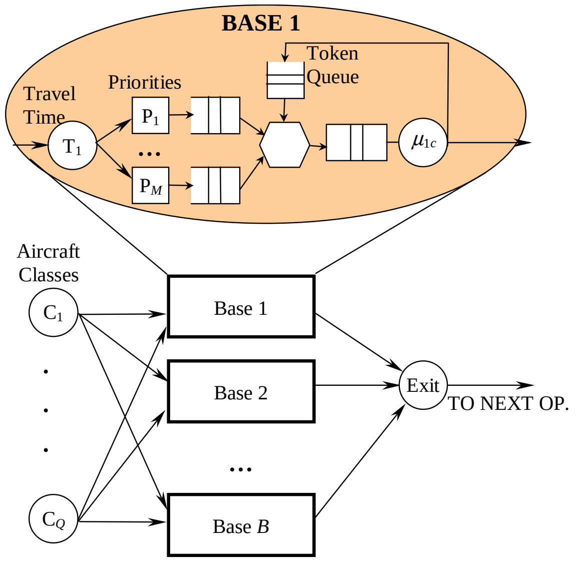


Figure C.5: ARMS Model

C.3.1 Simulation Results

To test the clustering ideas we use the following simple problem. We assume that this operation involves $A = 100$ aircraft that can be classified in $Q = 3$ classes. An aircraft is classified as C_1 with probability 0.25, as C_2 with probability 0.25 and as C_3 with probability 0.5. Each aircraft is routed probabilistically to one of two identical bases ($B = 2$). Each base has a total of 3 tokens and each class C_1, C_2, C_3 requires service for a random period of time which is Erlang distributed with means 1.2, 1.8, 2.4 time units respectively. The routing to each base depends on the initial "state of the world" W (e.g., current weather conditions). We assume that W can only take two values 0 and 1 with probabilities 0.8 and 0.2 respectively. If $W = 0$, then the aircraft are routed to either Base 1 or Base 2 with equal probability. On the other hand, when $W = 1$, all aircraft are routed to Base 1.

We simulated the ARMS model 65,000 times and obtained the histogram shown in Figure C.6 for the time it takes to process all 100 aircraft. One observation is that the average time to process all 100 aircraft is about 135 time units, however, the probability of actually processing all aircraft in 135 hours is very small. From the simulation results we also obtain an estimate of the standard deviation which was found equal to 33.9 time units.

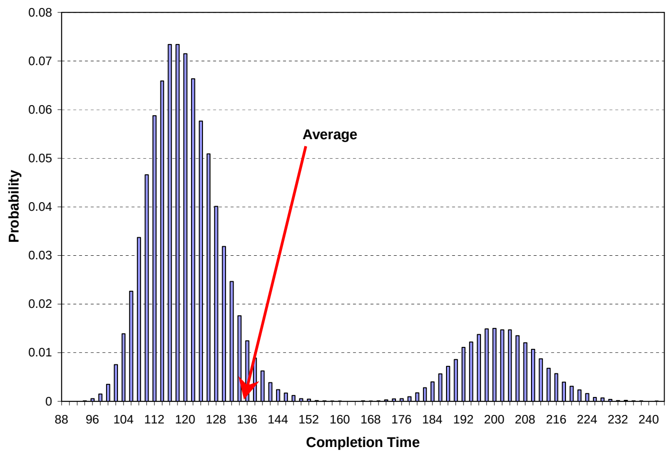


Figure C.6: Completion time of the first $A = 100$ aircraft

For planning purposes now, it is required to find the smallest deadline such that the probability of missing it is less that $P_0 = 10\%$. Using the normal distribution with $m = 135$ and $\sigma = 33.9$ we find using (C.1) that the deadline should be set at 178.4 time units. However, using **all** simulation results it is found that in order to meet the deadline with probability equal to 90% it is necessary to set it at 200 time units. Therefore, use of the simple averages has resulted in an error of about 11%. What is more dramatic is the error in the probability of missing the deadline. If we use the deadline suggested by the previous procedure on the data obtained through simulation, it is observed that the actual probability of missing the deadline is not 10% as was originally desired but about 19.7%, therefore there is an error of about 98%. What is also interesting is that this error was obtained even though we passed both, first and second order statistics to the low-resolution model. One might expect the errors to be even larger in cases where only first order statistics are passed to the low-resolution model.

To solve this problem, we can use a simple clustering approach. Rather than using a single average and standard deviation, we can form groups of data, determine the average and standard deviation of each group and use those estimates to drive the low-resolution model. For our simulation example it is natural to cluster the obtained data based on the initial conditions of each simulation. Therefore, all data that were obtained when $W = 0$ form cluster 0 and data that were obtained when $W = 1$ form cluster 1. Using this grouping, we obtain the following results

	Cluster 0	**Cluster 1**
Samples	52,014	12,986
m	118.5	200.3
σ	8.7	10.4

Using these results, we form the weighted sum of two Normal distributions

$$\acute{f}(x) = \frac{52,014}{65,000} N(m_0, \sigma_0) + \frac{12,986}{65,000} N(m_1, \sigma_1) \tag{C.2}$$

Finally, using trial and error we find that in order to meet the deadline with probability 10%, it is necessary to set the deadline at 200 time units which is in agreement with the simulation results as well.

If clustering is used as an approach to preserve statistical fidelity, we need a systematic way of grouping sample paths into clusters, especially when there is no apparent way to classify sample paths like the "world state" W we used in the above example. Such approaches are presented in the next two sections.

C.4 Clustering using Adaptive Resonance Theory (ART)

Our work is motivated by our earlier path bundle grouping approach in hierarchical combat simulation. In dealing with hierarchical simulation models one needs to consider grouping sample paths generated from high-resolution models so as to provide appropriate input statistics to the lower resolution model. This requires clustering very high dimensional data vectors (the sample paths from high-resolution simulators). We have used this approach in the Concept Evaluation Model (CEM) of the Concept Analysis Agency in order to group the sample paths from the high-resolution Combat Sample Generator (COSAGE) and generate the input to the lower resolution Attrition Calculation (ATCAL). Concrete numerical results are reported in Guo et. al[39].

The algorithm used for clustering in the ART framework is closely related to the well-known k-means clustering algorithm. Both use single prototypes to internally represent and dynamically adapt clusters. The k-means algorithm clusters a given set of input patterns into k groups. The parameter k thus specifies the coarseness of the partition. In contrast, ART uses a minimum required similarity between patterns that are grouped within one cluster. The resulting number of clusters depends on the distances (in terms of the applied metric) between all input patterns, presented to the network during training cycles. This similarity parameter is called vigilance and is denoted by ρ.

The basic ART architecture consists of the input layer F_1, the output/cluster layer F_2 and the reset mechanism which controls the degree of similarity required among patterns that are placed in the same cluster. F_1 has n input units, thus each input pattern must have dimension n, while the output layer consists of m units, therefore the maximum number of clusters that can be generated by the network is m. Note that F_2 is a competitive layer, in other words only the unit with the largest input has an activation other than zero, and therefore each unit corresponds to a different cluster. Furthermore, the input layer F_1 is subdivided into two sub-layers $F_1(a)$ the input portion and $F_1(b)$ the interface portion which receives input from both the input layer $F_1(a)$ and the output layer F_2.

In the ART algorithm, every input pattern, after some preprocessing, is compared to each of the m prototypes which are stored in the network's weights. If the degree of similarity between the current input pattern I and the best fitting prototype J is at least as high as a given vigilance ρ, prototype J is chosen to represent the cluster containing I; The vigilance ρ defines the minimum similarity between an input pattern and the prototype of the cluster it is associated with and is typically limited to the range $[0, 1]$. If the similarity between I and the best fitting prototype J does not fit into the vigilance interval $[\rho, 1]$, then a new cluster has to be installed, where the current input is most commonly used as the first prototype or "cluster center". Otherwise, if one of the previously committed clusters matches I well enough, its prototype is adapted by being slightly shifted towards the values of the input pattern I. For a detailed description and analysis of the ART algorithm refer to Carpenter and Grossberg [10] and Fausett [27].

Often, the level of detail of the input patterns may be different; some input patters may have less than n non-zero components. In ART this has motivated the use of a *similarity measure* that is independent of the magnitude of the input pattern and so input patterns are first normalized before presented to the neural network. Rather, the similarity measure is based on the *angle* between the vector that corresponds to an input pattern and the cluster's prototype vector as illustrated in Figure C.7 for a two-dimensional input vector ($n = 2$). This figure shows the jth cluster prototype vector $\mathbf{W}_j$ and the range of the cluster which extends β radians above and below the angle of $\mathbf{W}_j$. Note that β is a function of the vigilance parameter ρ; the larger the value of ρ the higher similarity is required among the vectors that are clustered in the same cluster and thus, the smaller the angle β. For this example, the angle between the input pattern I and $\mathbf{W}_j$ is $\phi > \beta$ therefore I will not be assigned to cluster j.

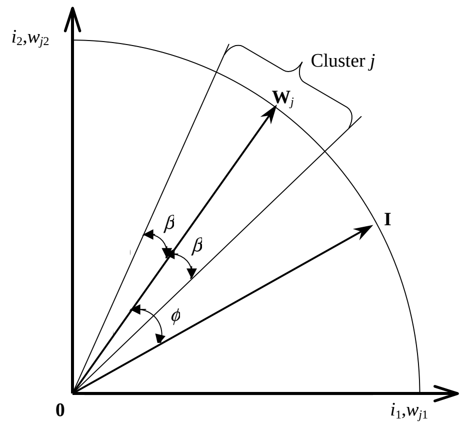


Figure C.7: Similarity in ART is measured by the angle

C.5 An application to a “real-world” complex system

As mentioned earlier, we are also developing a generic clustering tool, the Intelligent Clustering Interface (ICI) and we encountered an interesting opportunity to test it in the case of a complex manufacturing system. In particular, in working with a large metal manufacturer we were faced with the issue of supplying a low-resolution model of a large plant with the necessary parameters for running it. These parameters are to be obtained from detailed (high-resolution) models of the process plans (or flowpaths) for over 10,000 products manufactured in the plant. A flowpath is a specific sequence of Production Centers (PCs) with different processing characteristics at each PC (there are over 100 such PCs). Thus, each flowpath may be thought of as corresponding to a unique product; however, since the low-resolution model cannot possibly handle input data for over 10,000 flowpaths, the objective is to group products with similar flowpaths. For purposes such as forecasting, capacity planning, and lead-time estimation (among others) it is in fact indispensable to have such product groups available: not only it is conceptually infeasible to work with over 10,000 distinct products, it is also practically impossible to input such high-dimensional data for over 10,000 products and 100 PCs into modeling and decision support tools. Moreover, even if there were an automated way to accomplish this, it would be unrealistic to expect anyone to manipulate or interpret output data with information such as inventory levels and lead times for many thousands of distinct products.

In the effort to establish groups (or clusters) of products based on similarities in flowpaths and processing characteristics, an initial project was set up with plant experts given the task to “manually” create such groupings. The project was quickly abandoned: in addition to the sheer product volume which makes this task prohibitive, it is also difficult to rationally quantify “similarities” in flowpaths and processing data without some systematic means of doing so. We were able to accomplish this task using the clustering techniques we have developed and obtained a “compression” of over 10,000 products to 25-100 product clusters (depending on the aggregation accuracy required, which is completely controlled by the analyst). Of particular interest is the fact that the plant experts who reviewed the results we obtained found “by hindsight” the clusters defined by our method consistent with their expectations.

Unlike other applications of ART, in this case, the *magnitude* of the vector corresponding to an input pattern contains important information that should be used when grouping products into clusters. For example, the input vectors corresponding to two products may have the same orientation (same flowpath) but differ in their magnitude (the time spent at PC’s differ by orders of magnitude). For this reason we developed an enhancement to the ART algorithm that allows us to include magnitude information in the clustering process as described next.

C.5.1 ART Enhancements

As already pointed out, the basic mechanism through which the ART neural network performs clustering is by grouping them using an “angle” criterion. This is illustrated in

Figure C.8 where we used the ICI tool to cluster 200 2-dimensional vectors. For a vigilance parameter value of $\rho = 0.99$, three clusters were obtained as seen in the figure.

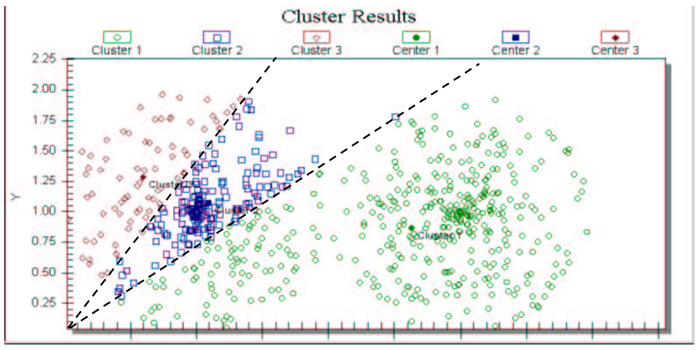


Figure C.8: 200 2-dim. vectors, no extra dimension, $\rho = 0.99$

It is reasonable to expect, however, that data vectors with almost identical orientation but significantly different magnitudes (prior to normalization) should be distinguishable. In order to introduce this capability into the ART setting, we have introduced the following enhancement: Each data vector is provided with an additional component (thus enlarging its dimensionality from n to $n+1$) which is the Euclidian norm of the n-dimensional vector $(x_1, \ldots, x_n)$, i.e., $(x_1^2 + \ldots + x_n^2)^{1/2}$. This forces the ART neural network to include magnitude information in its clustering algorithm. This is clearly seen in Figure C.9, where the same data vectors as in Figure C.8 have been clustered with this additional component. There are still three clusters, but one can see that the contents and features of the clusters have changed.

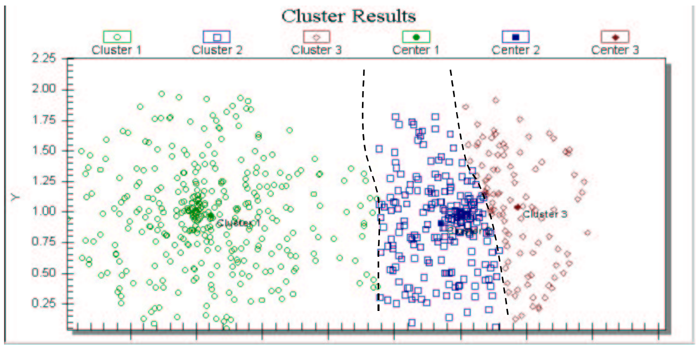


Figure C.9: Same 200 2-dim. vectors, WITH extra dimension, $\rho = 0.99$

An alternative is to apply the same idea by individually reclustering each of the clusters originally obtained. Thus, within each cluster we may now distinguish between data vectors in terms of their magnitude, as illustrated in Figure C.10 where 10 clusters are now obtained by reclustering each of the three clusters in Figure C.8 using a magnitude component.

Finally, note that within the ICI tool there is the capability of providing a “weight” to each component of the n-dimensional data vectors. Thus, by controlling the value of this

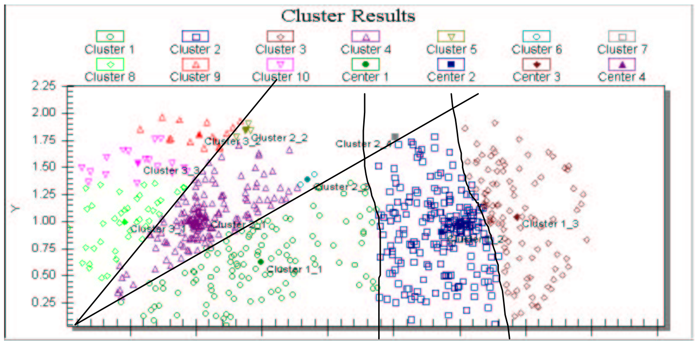


Figure C.10: Same 200 2-dim. vectors, WITH extra dim. reclustered within each of the 3 original clusters, $\rho = 0.99$

weight for the extra component added, we can adjust the importance to be attributed to the magnitude of a vector as opposed to just its orientation in n dimensions.

C.6 Using Hidden Markov Model for Sample Path Clustering

A sample path generated by a discrete-event system consists of a sequence $\{(e_k, t_k)\}$, $k = 1, 2, \cdots, K$, where e_k denotes the kth event and t_k its corresponding occurrence time. For typical systems, the number of observed events K is very large and thus clustering sample paths "directly" by making explicit use of the entire event sequence $\{(e_k, t_k)\}$, requires that the input vectors corresponding to such sample paths should be of a very large dimension. This impose a significant computational burden on any clustering algorithm including ART. To solve this problem we try to take advantage of the structure of discrete-event sample paths. In our experiments, we observe a sample path that is generated by an arbitrary system and try to describe it by some Markov Chain, thus we use the theory of Hidden Markov Models [69] (HMMs) to identify its parameters. Once we identify the HMM parameters we define a *similarity measure* among each obtained HMM and cluster together all sample paths with the largest similarity. The advantage of this approach is that the amount of information required to describe an HMM is considerably less than the amount of information required to describe a sample path. Even though the identification of the HMM parameters require some additional computational overhead, our experiments have shown that overall, the HMM approach is considerably faster than direct clustering approaches. Incidentally, we point out that this approach makes no a priori assumptions about the statistical distribution of the data to be analyzed.

Next, we demonstrate the HMM clustering approach through an example. For the purposes of our example, we assume that we have three systems S_1, S_2 and S_3. When simulated, each system generates sample paths Q_{ij}, $i = \{1, 2, 3\}$, $j = 1, 2, \cdots$ where Q_{ij} corresponds to the jth sample path generated by system S_i. When clustering sample paths, it would be reasonable to expect that sample paths generated from the same system are grouped in the same cluster. In this example, we generate 9 sample paths, 3 from each system and develop a way to distinguish between sample paths obtained from different systems. To achieve this, we first associate an HMM $\lambda_i = (A_i, B_i, \pi_i)$ to each sample path $i = 1, \cdots, 9$, where we use the notation of Rabiner[69]. A_i denotes the state transition probability, B_i denotes the observation symbol probability at every state and π_i denotes the initial state distribution. To complete the definition of the HMM, we assume the it consists of N states and that at every state we can observe any of the M possible symbols.

To construct the three systems S_i, $i = 1, 2, 3$ we assume that they consist of a Markov Chain with N_i states ($N_1 = 20$, $N_2 = 10$ and, $N_3 = 10$) and randomly generate a state transition probability matrix $\mathbf{P}_i = [p^i_{kl}]$, $k, l = 1, 2, \cdots, N_i$. Furthermore, we randomly generate the parameters μ^i_k, $k = 1, 2, \cdots, N_i$, so that the exponentially distributed sojourn time for state k has mean $1/\mu^i_k$. However, not all real systems are memoryless, therefore, to make our example more interesting we introduce some "special states" where state transitions out of such states are not made according to the state transition probability but rather through the set of rules we describe next.

For S_1, we assume that states $1, \cdots, 5$ are special states and state transitions out of these states are made according to the following rules:

State 1: System stays at state 1 for 3 consecutive, exponentially distributed sojourn time intervals, each with mean $1/\mu^1_1$. Then it jumps to state 10.

State 2: System stays at state 2 a deterministic sojourn time interval of length $2/\mu^1_2$. Then it jumps to state n according to the state visited before arriving at state 2, denoted by S_{-1}:

$$n = \begin{cases} 15 & \text{if } S_{-1} \in \{1, \cdots, 10\}, \\ 4 & \text{if } S_{-1} \in \{10, \cdots, 20\}. \end{cases}$$

State 3: System stays at state 3 for 5 consecutive sojourn time intervals, each exponentially distributed with mean $1/\mu^1_3$. Then it transfers according to state transition probability matrix $\mathbf{P}_1$. Note that after the system has spent 5 sojourn intervals at state 3, it may return to state 3 with probability p^1_{33} for 5 more intervals.

State 4: System stays at state 4 for 1 exponentially distributed sojourn time interval with mean $1/\mu^1_4$. Then it jumps to state n according to the state visited before arriving at state 4:

$$n = \begin{cases} 6 & \text{if } S_{-1} \in \{1, \cdots, 5\}, \\ 11 & \text{if } S_{-1} \in \{6, \cdots, 10\}, \\ 16 & \text{if } S_{-1} \in \{11, \cdots, 15\}, \\ 1 & \text{if } S_{-1} \in \{16, \cdots, 20\}, \end{cases}$$

State 5: System stays at state 5 for a deterministic sojourn interval of length $1/\mu^1_5$, and then transfers according to the state transition probability matrix $\mathbf{P}_1$.

Both S_2 and S_3 have only 2 special states, states 1 and 2 and the rules out of these states are the following:

State 1: System stays at state 1 for 2 sojourn time interval, each of which is exponentially distributed with mean $1/\mu_1^i$, $i = 2, 3$. Then it transfers to state n according to the previous state visited:

$$n = \begin{cases} 7 & \text{if } S_{-1} \in \{1, 2\}, \\ 9 & \text{if } S_{-1} \in \{3\}, \\ 5 & \text{if } S_{-1} \in \{4, \cdots, 10\}, \end{cases}$$

State 2: System stays at state 2 for a deterministic amount of time equal to $1/\mu_2^i$, $i = 2, 3$, and then transfers to state 5.

For each of the 9 sample paths we generate by S_1, S_2, and S_3, we adjust the HMM parameters λ_j, $j = 1, \cdots, 9$, to maximize the probability that the jth observed sample path was obtained from λ_j. This is referred to as the *training* problem and is tackled by repeatedly solving what is described as "Problem 3" in Rabiner[69]. For the purposes of this experiment, we assume that each HHM consists of $N = 6$ states. In addition, we assumed that the actual state visited by each of the systems is not observable. Rather, the observation symbols at each state are the state holding times. These can generally take any positive value so to determine B_i, the observed symbol probability, we quantize all possible values into $M = 64$ intervals.

Once we determine the HMM parameters for all sample paths, λ_i, $i = 1, \cdots, 9$, we use the *similarity measure* also defined in Rabiner[69] to determine which HMMs and consequently which sample paths are sufficiently similar so that they can be clustered together. The similarity measure is defined for any pair of HMMs λ_i and λ_j as:

$$\sigma(\lambda_i, \lambda_j) = \exp\{D(\lambda_i, \lambda_j)\} \tag{C.3}$$

where

$$D(\lambda_i, \lambda_j) = \frac{\log \Pr(O^j|\lambda_i) + \log \Pr(O^i|\lambda_j) - \log \Pr(O^i|\lambda_i) - \log \Pr(O^j|\lambda_j)}{2TK} \tag{C.4}$$

is what Rabiner [69] called the *distance measure*. $\Pr(O^i|\lambda_j)$ is the probability of the observation sequence O^i, i.e., the sequence of state holding times that correspond to the sample path Q_i, was generated by HMM λ_j. For computational convenience, we break any sample path into K smaller sample paths of length T and thus compute

$$\log \Pr\left(O^i|\lambda_j\right) = \sum_{k=1}^{K} \log \Pr\left(O_k^i|\lambda_j\right)$$

where $\Pr\left(O_k^i|\lambda_j\right)$ is the probability of the kth sequence of sample path i was generated by HMM λ_i. Also, note that the similarity measure is symmetric, that is $\sigma(\lambda_i, \lambda_j) = \sigma(\lambda_j, \lambda_i)$; a desired property for a good similarity measure.

For the similarity results shown in Table C.1, the length of each of the 9 sample path is 10,000 events. In addition, the parameters μ_j^i, $i = 1, 2, 3$, $j = 1, \cdots, N_i$ are generated such

Table C.1: Similarity measure among the HMMs corresponding to each of the 9 sample paths.

$\sigma(i,j)$	HMM 1	HMM 2	HMM 3	HMM 4	HMM 5	HMM 6	HMM 7	HMM 8	HMM 9
HMM 1	1	0.760	0.769	0.776	**0.950**	**0.950**	0.798	0.804	0.794
HMM 2	0.760	1	**0.940**	**0.949**	0.772	0.776	0.837	0.839	0.835
HMM 3	0.769	**0.940**	1	**0.947**	0.777	0.785	0.850	0.847	0.847
HMM 4	0.776	**0.949**	**0.947**	1	0.787	0.793	0.847	0.844	0.844
HMM 5	**0.950**	0.772	0.777	0.787	1	**0.951**	0.799	0.804	0.799
HMM 6	**0.950**	0.776	0.785	0.793	**0.951**	1	0.815	0.820	0.809
HMM 7	0.798	0.837	0.850	0.847	0.799	0.815	1	**0.945**	**0.943**
HMM 8	0.804	0.839	0.847	0.844	0.804	0.820	**0.945**	1	**0.950**
HMM 9	0.794	0.835	0.847	0.844	0.799	0.809	**0.943**	**0.950**	1

that $1/\mu_j^i$ are uniformly distributed between 4 and 50. Sample paths Q_1, Q_5 and Q_6 are generated by S_1. Q_2, Q_3, and Q_4 are generated by S_2 while Q_7, Q_8 and Q_9 are generated by S_3.

Finally, we cluster together all sample paths that correspond to HMMs with similarity measure greater than a threshold value V. Note that V corresponds to the required degree of similarity for two sample paths to be clustered together, like the vigilance parameter ρ of ART. For example, if $V = 0.9$, then the similarity measures exceeding V are:

Cluster 1: $\sigma(1,5)$, $\sigma(1,6)$, $\sigma(5,6)$
Cluster 2: $\sigma(2,3)$, $\sigma(2,4)$, $\sigma(3,4)$
Cluster 3: $\sigma(7,8)$, $\sigma(7,9)$, $\sigma(8,9)$

therefore, the HMM method has successfully classified all sample paths.

C.7 Conclusion

In this paper, we discuss the basic concepts involved in multi-resolution simulation modeling of complex stochastic systems and demonstrate that, using the average over all available high-resolution simulation results as the input to subsequent low-resolution modules is inappropriate and can lead to erroneous final results. Instead high-resolution output data should be clustered into groups that match underlying features of the system behavior before feeding group averages to low-resolution modules. In addition, we propose two approaches for performing high-dimensional data clustering based on neural networks and Hidden Markov Models.

Appendix D

A Generalized ‘Surrogate Problem’ Methodology for On-Line Stochastic Discrete Optimization

Kagan Gokbayrak and Christos G. Cassandras[1]

Department of Manufacturing Engineering, Boston University, Boston, MA 02215

Abstract

We consider stochastic discrete optimization problems where the decision variables are non-negative integers and propose a generalized “surrogate problem” methodology that modifies and extends previous work in [33]. Our approach is based on an *on-line* control scheme which transforms the problem into a “surrogate” continuous optimization problem and proceeds to solve the latter using standard gradient-based approaches while simultaneously updating both actual and surrogate system states. In contrast to [33], the proposed methodology applies to arbitrary constraint sets. It is shown that, under certain conditions, the solution of the original problem is recovered from the optimal surrogate state. Applications of this approach include solutions to multicommodity resource allocation problems, where, exploiting the convergence speed of the method, one can overcome the obstacle posed by the presence of local optima.

[1]This work was supported in part by the National Science Foundation under Grants EEC-95-27422 and ACI-98-73339, by AFOSR under contract F49620-98-1-0387, by the Air Force Research Laboratory under contract F30602-99-C-0057 and by EPRI/ARO under contract WO8333-03.

We consider stochastic discrete optimization problems where the decision variables are non-negative integers. Problems of this type abound, for instance, in manufacturing systems and communication networks. In a manufacturing system setting, examples include the classic buffer allocation problem, where K buffer slots are to be distributed over N manufacturing workstations so as to optimize performance criteria involving throughput or mean system time; a variant of this problem involving the use of kanban (rather than buffer slots) to be allocated to different workstations [21]; and determining the optimal lot size for each of N different part types sharing resources in a production facility with setup delays incurred when a switch from a lot of one part type to another occurs [40]. In a communication network setting, similar buffer allocation issues arise, as well as transmission scheduling problems where a fixed number of time slots forming a "frame" must be allocated over several call types [6]. Such optimization problems are also very common in any discrete resource allocation setting [44], as well as in control policies for Discrete Event Systems (DES) that are parameterized by discrete variables such as *thresholds* or *hedging points.*

The optimization problem we are interested in is of the general form

$$\min_{r \in A_d} J_d(r) = E[L_d(r, \omega)] \tag{D.1}$$

where $r \in \mathbb{Z}_+^N$ is a decision vector or "state" and A_d represents a constraint set. In a stochastic setting, let $L_d(r, \omega)$ be the cost incurred over a specific sample path ω when the state is r and $J_d(r) = E[L_d(r, \omega)]$ be the expected cost of the system operating under r. The sample space is $\Omega = [0, 1]^\infty$, that is, $\omega \in \Omega$ is a sequence of random numbers from $[0, 1]$ used to generate a sample path of the system. The cost functions are defined as $L_d : A_d \times \Omega \to \mathbb{R}$ and $J_d : A_d \to \mathbb{R}$, and the expectation is defined with respect to a probability space $(\Omega, \Im, P)$ where $\Im$ is an appropriately defined σ-field on Ω and P is a conveniently chosen probability measure. In the sequel, 'ω' is dropped from $L_d(r, \omega)$ and, unless otherwise noted, all costs will be over the same sample path.

The problem (D.1) is a notoriously hard stochastic integer programming problem. Even in a deterministic setting, where $J_d(r) = L_d(r)$, this class of problems is NP-hard (see [44], [68] and references therein). In some cases, depending upon the form of the objective function $J_d(r)$ (e.g., separability, convexity), efficient algorithms based on finite-stage dynamic programming or generalized Lagrange relaxation methods are known (see [44] for a comprehensive discussion on aspects of deterministic resource allocation algorithms). Alternatively, if no a priori information is known about the structure of the problem, then some form of a search algorithm is employed (e.g., Simulated Annealing [1], Genetic Algorithms [43]).

When the system operates in a stochastic environment (e.g., in a resource allocation setting, users request resources at random time instants or hold a particular resource for a random period of time) and no closed-form expression for $E[L_d(r)]$ is available, the problem is further complicated by the need to estimate $E[L_d(r)]$. This generally requires Monte Carlo simulation or direct measurements made on the actual system. Most known approaches are based on some form of random search, as in algorithms proposed by Yan and Mukai [87], Gong et al [36], Shi and Olafsson [73]. Another recent contribution to this area involves the *ordinal* optimization approach presented in [42] and used in [18] to solve a class of resource allocation problems. Even though the approach in [18] yields a fast algorithm, it is still constrained to iterate so that every step involves the transfer of no more than a single resource from one user to some other user. One can expect, however, that much faster improvements can be realized in a scheme allowed to reallocate multiple resources from users whose cost-sensitivities are small to users whose sensitivities are much larger. This is precisely the rationale of most gradient-based continuous optimization schemes, where the gradient is a measure of this sensitivity.

With this motivation in mind, a new approach was proposed in [33] based on the following idea: The *discrete* optimization problem (D.1) is transformed into a "surrogate" *continuous* optimization problem which is solved using standard gradient-based methods; its solution is then transformed back into a solution of the original problem. Moreover, this process is designed explicitly for *on-line* operation. That is, at every iteration step in the solution of the surrogate continuous optimization problem, the surrogate continuous state is immediately transformed into a feasible discrete state r. This is crucial, since whatever information is used to drive the process (e.g., sensitivity estimates) can only be obtained from a sample path of the *actual* system operating under r. It was shown in [33] that for resource allocation problems, where the constraint set is of the form $A_d = \left\{ r : \sum_{i=1}^{N} r_i = K \right\}$, the solution of (D.1) can be recovered from the solution of the surrogate continuous optimization problem; the latter is obtained using a stochastic approximation algorithm which converges under standard technical conditions.

The contributions of this paper are the following. First, we generalize the methodology presented in [33] to problems of the form (D.1) which are not necessarily limited to constraints such as $A_d = \left\{ r : \sum_{i=1}^{N} r_i = K \right\}$, including the possibility of unconstrained problems. Second, we modify the approach developed in [33] in order to improve its computational efficiency. In particular, computational efficiency is gained in the following respects:

1. A crucial aspect of the "surrogate problem" method is the fact that the surrogate state, denoted by $\rho \in \mathbb{R}_+^N$, can be expressed as a convex combination of at most $N+1$ points in A_d, where N is the dimensionality of $r \in A_d$. Determining such points is not a simple task. In [33], this was handled using the Simplex Method of Linear Programming, which can become inefficient for large values of N. In this paper, we show that for any surrogate state ρ, a selection set $S(\rho)$ of such $N+1$ points, *not necessarily in* A_d, can be identified through a simple algorithm of linear complexity. Moreover, this algorithm applies to any problem of the form (D.1), not limited to any special type of constraint set A_d.

2. In solving the surrogate continuous optimization problem, a surrogate objective function is defined whose gradient is estimated in order to drive a stochastic approximation type of algorithm. The gradient estimate computation in [33] involves the inversion of an $N \times N$ matrix. In this paper, we show that this is not needed if one makes use of the selection set $S(\rho)$ mentioned above, and the gradient estimate computation is greatly simplified.

The price to pay for the generalization of the approach is the difficulty in establishing a general result regarding the recovery of the solution of (D.1) from the solution of the surrogate problem as was done in our earlier work [33]. We are able, however, to still do so for two interesting cases. Despite this difficulty, the empirical evidence to date indicates that this generalized methodology provides the optimal solutions under appropriate technical conditions guaranteeing convergence of a stochastic approximation scheme.

A third contribution of this paper is in tackling a class of particularly hard *multicommodity* discrete optimization problems, where multiple local optima typically exist. Exploiting the convergence speed of the surrogate method, we present, as an application of the proposed approach, a systematic means for solving such combinatorially hard problems.

The rest of the paper is organized as follows. In Section D.1, we give an overview of our basic approach. In Section D.2, we present the key results enabling us to transform a discrete stochastic optimization problem into a "surrogate" continuous optimization problem. In Section D.3, we discuss the construction of appropriate "surrogate" cost functions for our approach and the evaluation of their gradients. Section D.4 discusses how to recover the solution of the original problem from that of the "surrogate" problem. In Section D.5, we present the detailed optimization algorithm and discuss its convergence properties. Some numerical examples and applications are presented in Section D.6.

D.1 Basic approach for on-line control

In the sequel, we shall adopt the following notational conventions as in [33]. We shall use subscripts to indicate components of a vector (e.g., r_i is the ith component of r). We shall use superscripts to index vectors belonging to a particular set (e.g., r^j is the jth vector within a subset of $\mathbb{Z}_+^N$ that contains such vectors). Finally, we reserve the index n as a subscript that denotes iteration steps and not vector components (e.g., r_n is the value of r at the nth step of an iterative scheme, not the nth component of r).

The expected cost function $J_d(r)$ is generally nonlinear in r, a vector of integer-valued decision variables, therefore (D.1) is a nonlinear integer programming problem. One common method for solving this problem is to relax the integer constraints on all r_i so that they can be regarded as continuous (real-valued) variables and then to apply standard optimization techniques such as gradient-based algorithms. Let the "relaxed" set A_c contain the original constraint set A_d and define $\bar{L}_c : \mathbb{R}_+^N \times \Omega \to \mathbb{R}$ to be the cost function over a specific sample path. As before let us drop 'ω' from $\bar{L}_c(\rho, \omega)$ and agree that unless otherwise noted all costs will be over the same sample path. The resulting "surrogate" problem then becomes: Find ρ^* that minimizes the "surrogate" expected cost function $J_c : \mathbb{R}_+^N \to \mathbb{R}$ over the continuous set A_c, i.e.,

$$J_c(\rho^*) = \min_{\rho \in A_c} J_c(\rho) = E[\bar{L}_c(\rho)] \tag{D.2}$$

where $\rho \in \mathbb{R}_+^N$, is a real-valued state, and the expectation is defined on the same probability space $(\Omega, \Im, P)$ as described earlier. Assuming an optimal solution ρ^* can be determined, this state must then be mapped back into a discrete vector by some means (usually, some form of truncation). Even if the final outcome of this process can recover the actual r^* in (D.1), this approach is strictly limited to *off-line* analysis: When an iterative scheme is used to solve the problem in (D.2) (as is usually the case except for very simple problems of limited interest), a sequence of points $\{\rho_n\}$ is generated; these points are generally continuous states in A_c, hence they may be infeasible in the original discrete optimization problem. Moreover, if one has to estimate $E[\bar{L}_c(\rho)]$ or $\frac{\partial E[\bar{L}_c(\rho)]}{\partial \rho}$ through simulation, then a simulation model of the surrogate problem must be created, which is also not generally feasible. If, on the other hand, the only cost information available is through direct observation of sample paths of an actual system, then there is no obvious way to estimate $E[\bar{L}_c(\rho)]$ or $\frac{\partial E[\bar{L}_c(\rho)]}{\partial \rho}$, since this applies to the real-valued state ρ, not to the integer-valued actual state r.

As in [33], we adopt here a different approach intended to operate *on line*. In particular, we still invoke a relaxation such as the one above, i.e., we formulate a surrogate continuous optimization problem with some state space $A_c \subset \mathbb{R}_+^N$ and $A_d \subset A_c$. However, at every step n of the iteration scheme involved in solving the problem, both the continuous and the discrete states are simultaneously updated through a mapping of the form $r_n = f_n(\rho_n)$. This has two advantages: First, the cost of the original system is continuously adjusted (in contrast to an adjustment that would only be possible at the end of the surrogate minimization process); and second, it allows us to make use of information typically needed to obtain cost sensitivities from the actual operating system at every step of the process.

The basic scheme we consider is the same as in [33] and is outlined below for the sake of self-sufficiency of the paper. Initially, we set the "surrogate system" state to be that of the actual system state, i.e.,

$$\rho_0 = r_0 \tag{D.3}$$

Subsequently, at the nth step of the process, let $H_n(\rho_n, r_n, \omega_n)$ denote an estimate of the sensitivity of the cost $J_c(\rho_n)$ with respect to ρ_n obtained over a sample path ω_n of the actual system operating under allocation r_n; details regarding this sensitivity estimate will be provided later in the paper. Two sequential operations are then performed at the nth step:

1. The continuous state ρ_n is updated through

$$\rho_{n+1} = \pi_{n+1}[\rho_n - \eta_n H_n(\rho_n, r_n, \omega_n)] \tag{D.4}$$

   where $\pi_{n+1} : \mathbb{R}^N \to A_c$ is a projection function so that $\rho_{n+1} \in A_c$ and η_n is a "step size" parameter.

2. The newly determined state of the surrogate system, ρ_{n+1}, is transformed into an actual feasible discrete state of the original system through

$$r_{n+1} = f_{n+1}(\rho_{n+1}) \tag{D.5}$$

   where $f_{n+1} : A_c \to A_d$ is a mapping of feasible continuous states to feasible discrete states which must be appropriately selected as will be discussed later.

One can recognize in (D.4) the form of a stochastic approximation algorithm (e.g., [50]) that generates a sequence $\{\rho_n\}$ aimed at solving (D.2). However, there is an additional operation (D.5) for generating a sequence $\{r_n\}$ which we would like to see converge to r^* in (D.1). It is important to note that $\{r_n\}$ corresponds to feasible realizable states based on which one can evaluate estimates $H_n(\rho_n, r_n, \omega_n)$ from observable data, i.e., a sample path of the actual system under r_n (not the surrogate state ρ_n). We can therefore see that this scheme is intended to combine the advantages of a stochastic approximation type of algorithm with the ability to obtain sensitivity estimates with respect to discrete decision variables. In particular, sensitivity estimation methods for discrete parameters based on Perturbation Analysis (PA) and Concurrent Estimation [41],[20] are ideally suited to meet this objective.

Before addressing the issue of obtaining estimates $H_n(\rho_n, r_n, \omega_n)$ necessary for the optimization scheme described above to work, there are two other crucial issues that form the cornerstones of the proposed approach. First, the selection of the mapping f_{n+1} in (D.5) must be specified. Second, a surrogate cost function $\bar{L}_c(\rho, \omega)$ must be identified and its relationship to the actual cost $L_d(r, \omega)$ must be made explicit. These issues are discussed next, in Sections D.2 and D.3 respectively for the problem (D.1), which, as previously mentioned, is not limited to the class of resource allocation problems considered in our earlier work [33].

D.2 Continuous-to-discrete state transformations

Let us first define $\mathcal{C}(\rho)$, the set of vertices of the unit "cube" around the surrogate state as

$$\mathcal{C}(\rho) = \{r | \forall i \; r_i \in \{\lfloor \rho_i \rfloor, \; \lceil \rho_i \rceil\}\}$$

where, for any $x \in \mathbb{R}$, $\lceil x \rceil$ and $\lfloor x \rfloor$ denote the ceiling (smallest integer $\geq x$) and floor (largest integer $\leq x$) of x respectively. Note that when $\rho_i \in \mathbb{Z}$, all the ith components of the cube elements are the same ($= \rho_i$) decreasing the dimension of the cube by one. In order to avoid the technical complications due to integer components in ρ, let us agree that whenever this is the case we will perturb the integer components to obtain a new state $\hat{\rho}$ whose components are non-integer, and then relabel this state as ρ.

Next, we define $\mathcal{N}(\rho)$, the set of all *feasible neighboring discrete states* in $\mathcal{C}(\rho)$ as:

$$\mathcal{N}(\rho) = \mathcal{C}(\rho) \cap A_d \tag{D.6}$$

A more explicit and convenient characterization of the set $\mathcal{N}(\rho)$ is

$$\mathcal{N}(\rho) = \{r | r = \lfloor \rho \rfloor + \tilde{r} \text{ for all } \tilde{r} \in \{0,1\}^N\} \cap A_d$$

where $\lfloor \rho \rfloor$ is the vector whose components are $\lfloor \rho \rfloor_i = \lfloor \rho_i \rfloor$. In other words, $\mathcal{N}(\rho)$ is the set of vertices of the unit "cube" containing ρ that are in the feasible discrete set A_d.

In earlier work (see [33]), we limited ourselves to resource allocation problems with linear capacity constraints. For this class of problems, we used $A_c = conv(A_d)$ as the feasible set in the "surrogate" continuous state space. When the feasible set A_d is not a polyhedron, the set $A_c = conv(A_d)$ may include discrete states that are not in A_d. In order to prevent this and to generalize the approach, we modify the definition of A_c, given the set $\mathcal{N}(\rho)$, as follows:

$$A_c = \bigcup_{\rho \in \mathbb{R}_+^N} conv(\mathcal{N}(\rho)) \tag{D.7}$$

Note that $A_c \subseteq conv(A_d)$ is the union of the convex hulls of feasible discrete points in every cube, and is not necessarily convex. Note also that the definition reduces to $A_c = conv(A_d)$ when A_d is formed by all the discrete points in a polyhedron.

Now we are ready to define the set of *transformation functions* $\mathcal{F}_\rho$ as follows:

$$\mathcal{F}_\rho = \{f | f : A_c \to A_d, \quad (f(\rho))_i \in \{\lceil \rho_i \rceil, \lfloor \rho_i \rfloor\}, \ i = 1, \ldots, N\}$$

The purpose of $f \in \mathcal{F}_\rho$ is to transform some continuous state vector $\rho \in A_c$ into a "neighboring" discrete state vector $r \in \mathcal{N}(\rho)$ obtained by seeking $\lceil \rho_i \rceil$ or $\lfloor \rho_i \rfloor$ for each component $i = 1, \ldots, N$. The existence of such a transformation is guaranteed by the projection mapping π in (D.4), which ensures that $\rho \in A_c$, therefore $\mathcal{N}(\rho)$ is non-empty. A convenient element of $\mathcal{F}_\rho$ that we shall use throughout the paper is

$$f(\rho) = \arg \min_{r \in \mathcal{N}(\rho)} \|\rho - r\|$$

which maps the surrogate state ρ to the closest feasible neighbor in $\mathcal{N}(\rho)$. However, our analysis is certainly not limited to this choice.

A key element of our approach is based on the fact that ρ can be expressed as a convex combination of *at most* $N+1$ points in $\mathcal{C}(\rho)$, as shown in Theorem 11 below. Given that the cardinality of $\mathcal{C}(\rho)$ is combinatorially explosive, i.e., 2^N, determining the set of these points is not a simple task. In [33], it was shown that such a set of feasible points, $\mathcal{N}_N(\rho)$, a subset of $\mathcal{N}(\rho)$, can be determined using the Simplex Method when problems of the form (D.1) are limited to constraint sets $A_d = \left\{ r : \sum_{i=1}^N r_i = K \right\}$. In what follows, we provide a different approach based on defining a *selection set* $\mathcal{S}(\rho)$ which (a) allows us to specify the $N+1$ points in $\mathcal{C}(\rho)$ that define a set whose convex hull includes ρ for problems with arbitrary A_d, (b) is much simpler than the Simplex Method, and (c) simplifies the gradient estimation procedure as we will see in Section D.3. An important distinction between $\mathcal{N}_N(\rho)$ and $\mathcal{S}(\rho)$ is that the latter is not limited to include only feasible points $r \in \mathcal{N}(\rho)$.

Definition 1 *The set $\mathcal{S}(\rho) \subseteq \mathcal{C}(\rho)$ is a selection set if it satisfies the following conditions:*

- $|\mathcal{S}(\rho)| = N + 1$
- The surrogate state ρ resides in the convex hull of $\mathcal{S}(\rho)$, i.e., there exists $\{\alpha_i\}$ such that

$$\rho = \sum_{i=0}^N \alpha_i r^i, \quad \text{with} \quad \sum_{i=0}^N \alpha_i = 1, \quad \alpha_i \geq 0, \quad r^i \in \mathcal{S}(\rho)$$

- The vectors in the set $\left\{\bar{r}^i | \bar{r}^i = \left[\begin{array}{cc} 1 & r^i \end{array}\right],\ \ r^i \in \mathcal{S}(\rho)\right\}$ are linearly independent.

Next we will show the existence of the selection set $\mathcal{S}(\rho)$ by a constructive proof.

Theorem 11 *A selection set $\mathcal{S}(\rho)$ exists for any $\rho \in \mathbb{R}_+^N$.*

Proof. We construct a selection set $\mathcal{S}(\rho)$ for $\rho = [\rho_1, ..., \rho_N]$ as described below and prove that it satisfies all three conditions in Definition 1.

Let us define e_i as the N-dimensional unit vector whose ith component is 1; the *residual vector* $\tilde{\rho} = \rho - \lfloor \rho \rfloor$; and the N-dimensional ordering vector o such that $o_k \in \{1, ..., N\}$, $k = 1, ..., N$, and o_k satisfies

$$\tilde{\rho}_{o_k} \leq \tilde{\rho}_{o_{k+1}} \text{ for } k = 1, ..., N-1$$

Note that the definition of $\tilde{\rho}$ implies that $0 < \tilde{\rho}_j < 1$ for all $j = 1, ..., N$. Next, we define

$$\tilde{r}^{o_l} = \sum_{k=l}^{N} e_{o_k} \tag{D.8}$$

and

$$\alpha_{o_l} = \left\{ \begin{array}{cc} \tilde{\rho}_{o_l} - \tilde{\rho}_{o_{l-1}} & l > 1 \\ \tilde{\rho}_{o_1} & l = 1 \end{array} \right. \geq 0 \tag{D.9}$$

It follows from (D.9) that we can write

$$\tilde{\rho}_{o_l} = \sum_{k=1}^{l} \alpha_{o_k} \tag{D.10}$$

and note that

$$\tilde{\rho}_{o_N} = \sum_{k=1}^{N} \alpha_{o_k} = \sum_{j=1}^{N} \alpha_j$$

Using (D.9) we have defined α_i for $i = 1, \ldots, N$. In addition, we now define

$$\alpha_0 = 1 - \sum_{i=1}^{N} \alpha_i = 1 - \tilde{\rho}_{o_N} > 0 \tag{D.11}$$

Similarly, (D.8) defines $\tilde{r}^i$ for $i = 1, \ldots, N$. In addition, we define

$$\tilde{r}^0 = \mathbf{0} \tag{D.12}$$

where $\mathbf{0} = [0...0]$ is the N-dimensional zero vector. Note that we can write

$$\tilde{\rho} = \sum_{l=1}^{N} \tilde{\rho}_{o_l} e_{o_l} \tag{D.13}$$

Combining (D.10) and (D.13) and changing the summation indices gives

$$\tilde{\rho} = \sum_{l=1}^{N} \sum_{k=1}^{l} \alpha_{o_k} e_{o_l} = \sum_{k=1}^{N} \sum_{l=k}^{N} \alpha_{o_k} e_{o_l} \tag{D.14}$$

Then, using (D.8) we get

$$\tilde{\rho} = \sum_{k=1}^{N} \alpha_{o_k} \tilde{r}^{o_k} = \sum_{j=1}^{N} \alpha_j \tilde{r}^j \tag{D.15}$$

Next, we define

$$r^j = \lfloor \rho \rfloor + \tilde{r}^j \tag{D.16}$$

Then, we can write

$$\begin{aligned} \sum_{j=0}^{N} \alpha_j r^j &= \sum_{j=0}^{N} \alpha_j \lfloor \rho \rfloor + \sum_{j=0}^{N} \alpha_j \tilde{r}^j \\ &= \lfloor \rho \rfloor \sum_{j=0}^{N} \alpha_j + \sum_{j=0}^{N} \alpha_j \tilde{r}^j \end{aligned}$$

Observing that $\sum_{j=0}^{N} \alpha_j = 1$ from (D.11), and that

$$\sum_{j=0}^{N} \alpha_j \tilde{r}^j = \sum_{j=1}^{N} \alpha_j \tilde{r}^j + \alpha_0 \tilde{r}^0 = \tilde{\rho}$$

from (D.12) and (D.15), it follows that

$$\sum_{j=0}^{N} \alpha_j r^j = \lfloor \rho \rfloor + \tilde{\rho} = \rho \tag{D.17}$$

i.e., the convex hull formed by $\mathcal{S}(\rho) = \{r^0, ..., r^N\}$, with r^i defined in (D.16), contains ρ. This satisfies the second condition in Definition 1. Moreover, from (D.8), (D.12), and (D.16), it is obvious that $|\mathcal{S}(\rho)| = N + 1$, satisfying the first condition as well.

It remains to show that the vectors $\begin{bmatrix} 1 & r^i \end{bmatrix}$ with r^i defined in (D.16) are linearly independent. Consider the matrix $\begin{bmatrix} e & \mathbf{R} \end{bmatrix}$ where $e = [1 \cdots 1]'$ is the $(N+1)-$ dimensional vector of $1'$s and $\mathbf{R}$ is the matrix whose rows are vectors from $\mathcal{S}(\rho)$ such that

$$\mathbf{R} = \begin{bmatrix} r^0 \\ r^{o_N} \\ \vdots \\ r^{o_1} \end{bmatrix}$$

Using (D.8), (D.12), and (D.16), one can write $\begin{bmatrix} 1 & r^{o_l} \end{bmatrix} - \begin{bmatrix} 1 & r^{o_{l+1}} \end{bmatrix} = \begin{bmatrix} 0 & e_{o_l} \end{bmatrix}$ for $l < N$ and $\begin{bmatrix} 1 & r^{o_N} \end{bmatrix} - \begin{bmatrix} 1 & r^0 \end{bmatrix} = \begin{bmatrix} 0 & e_{o_N} \end{bmatrix}$. Finally, note that

$$\begin{bmatrix} 1 & r^0 \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0} \end{bmatrix} + \sum_{i=1}^{N} \lfloor \rho_i \rfloor \begin{bmatrix} 0 & e_i \end{bmatrix}$$

Using these arguments one can show that the matrix $\begin{bmatrix} e & \mathbf{R} \end{bmatrix}$ can be transformed into the identity matrix of dimension $N+1$ by row operations. Therefore, $\begin{bmatrix} e & \mathbf{R} \end{bmatrix}$ is non-singular, i.e., the third condition of Definition 1 is satisfied. Moreover, the inverse of $\begin{bmatrix} e & \mathbf{R} \end{bmatrix}$, which will be needed during the gradient estimation part of our approach in the next section, can be evaluated to give:

$$\begin{bmatrix} e & \mathbf{R} \end{bmatrix}^{-1} = \begin{bmatrix} 1+\lfloor\rho_{o_N}\rfloor & \lfloor\rho_{o_{N-1}}\rfloor - \lfloor\rho_{o_N}\rfloor & \lfloor\rho_{o_{N-2}}\rfloor - \lfloor\rho_{o_{N-1}}\rfloor & \cdots & -\lfloor\rho_{o_1}\rfloor \\ \multicolumn{5}{c}{-\hat{e}_{N+1-\bar{o}_1} + \hat{e}_{N+2-\bar{o}_1}} \\ \multicolumn{5}{c}{\vdots} \\ \multicolumn{5}{c}{-\hat{e}_{N+1-\bar{o}_N} + \hat{e}_{N+2-\bar{o}_N}} \end{bmatrix} \tag{D.18}$$

where $\hat{e}_i$ is the $(N+1)-$dimensional unit vector whose ith component is 1 and $\bar{o}$ is the N-dimensional ordering vector such that $\bar{o}_k \in \{1, ..., N\}$ satisfying the relation

$$o_i = j \Leftrightarrow \bar{o}_j = i$$

One can also verify that $\begin{bmatrix} e & \mathbf{R} \end{bmatrix}^{-1}$ above is such that $\begin{bmatrix} e & \mathbf{R} \end{bmatrix}^{-1}\begin{bmatrix} e & \mathbf{R} \end{bmatrix} = \mathbf{I}$. ■

We stress that the selection set $\mathcal{S}(\rho)$ is *not unique*; however, given a selection set $\mathcal{S}(\rho)$, the α_i values are unique for $i = 0, ..., N$. There are clearly different ways one can construct $\mathcal{S}(\rho)$, including randomized methods. For instance, one can start out by randomly selecting the first element of the selection set from $\mathcal{C}(\rho)$ and then proceed through a scheme similar to the one used above.

The following is an algorithmic procedure for constructing $\mathcal{S}(\rho)$ as presented in Theorem 11:

- Initialize the index set $I = \{1, ..., N\}$ and define a temporary vector $v = \tilde{\rho}$.
- While $I \neq \emptyset$:

1. $\tilde{r}^i = \sum_{j\in I} e_j$ where $i = \arg\min\{v_j,\ j \in I\}$
2. $\alpha_i = v_i$
3. $v \leftarrow v - \alpha_i \tilde{r}^i$
4. $I \leftarrow I\backslash\{i\}$

- $\tilde{r}^0 = 0$
- $\alpha_0 = 1 - \sum_{i=1}^{N} \alpha_i$
- $\mathcal{S}(\rho) = \{r^i | r^i = \tilde{r}^i + \lfloor\rho\rfloor \text{ for } i = 0, ..., N\}$

Example: In order to clarify our notation and illustrate the specification of the sets $\mathcal{C}(\rho)$, $\mathcal{N}(\rho)$ and $\mathcal{S}(\rho)$, we provide the following example, which we will use throughout our analysis. Consider the allocation problem of $K = 10$ resources over $N = 3$ users, and let $\rho = [3.9, 3.9, 2.2]$. The feasible set is

$$A_c = \left\{ \rho : \sum_{i=1}^{N} \rho_i = 10 \right\} \tag{D.19}$$

Since $\lfloor \rho \rfloor = [3, 3, 2]$, we have the unit cube

$$\mathcal{C}(\rho) = \{[3,3,2], [3,3,3], [3,4,2], [3,4,3], [4,3,2], [4,3,3], [4,4,2], [4,4,3]\}$$

and the *feasible neighbors* of ρ are

$$\mathcal{N}(\rho) = \{[3,4,3], [4,4,2], [4,3,3]\}$$

Let us now construct a selection set satisfying the conditions of Definition 1 using the algorithm described above. First we initialize the index set $I = \{1, 2, 3\}$ and the residual vector $v = \rho - \lfloor \rho \rfloor = [0.9, 0.9, 0.2]$. Note that $\arg\min_{j \in \{1,2,3\}} \{v_j\} = 3$, therefore,

$$\begin{aligned} \tilde{r}^3 &= \sum_{j \in \{1,2,3\}} e_j = [1,1,1] \\ \alpha_3 &= v_3 = 0.2 \end{aligned}$$

Next, we update

$$\begin{aligned} v &\leftarrow v - \alpha_3 \tilde{r}^3 = [0.7, 0.7, 0] \\ I &\leftarrow I \backslash \{3\} = \{1, 2\} \end{aligned}$$

Now, note that the first two components in the updated v are equal. We can select any one of them for the next $\arg\min_{j \in \{1,2\}} \{v_j\}$. Thus, if we pick the first component we get

$$\begin{aligned} \tilde{r}^1 &= \sum_{j \in \{1,2\}} e_j = [1,1,0] \\ \alpha_1 &= v_1 = 0.7 \end{aligned}$$

Proceeding as before, we update

$$\begin{aligned} v &\leftarrow v - \alpha_1 \tilde{r}^1 = [0,0,0] \\ I &\leftarrow I \backslash \{1\} = \{2\} \end{aligned}$$

and finish this step by setting

$$\begin{aligned} \tilde{r}^2 &= \sum_{j \in \{2\}} e_j = [0,1,0] \\ \alpha_2 &= v_2 = 0 \end{aligned}$$

Finally,

$$\begin{aligned} v &\leftarrow v - \alpha_2 \tilde{r}^2 = [0,0,0] \\ I &\leftarrow I \backslash \{2\} = \{\} \end{aligned}$$

We may now construct the selection set from the vectors given by (D.16):

$$\begin{aligned}
r^1 &= \tilde{r}^1 + \lfloor \rho \rfloor = [1,1,0] + [3,3,2] = [4,4,2] \\
r^2 &= \tilde{r}^2 + \lfloor \rho \rfloor = [0,1,0] + [3,3,2] = [3,4,2] \\
r^3 &= \tilde{r}^3 + \lfloor \rho \rfloor = [1,1,1] + [3,3,2] = [4,4,3] \\
r^0 &= \tilde{r}^0 + \lfloor \rho \rfloor = [0,0,0] + [3,3,2] = [3,3,2]
\end{aligned}$$

i.e.,

$$\mathcal{S}(\rho) = \{[3,3,2], [3,4,2], [4,4,2], [4,4,3]\}$$

The example above illustrates the important difference between the selection set $\mathcal{N}_N(\rho)$ employed in [33] and the present construction: While all the elements of the set $\mathcal{N}_N(\rho)$ are feasible, the elements of the selection set $\mathcal{S}(\rho)$ constructed above may be infeasible. The following lemma considers resource allocation problems with total capacity constraints as a special case of discrete stochastic optimization and asserts that there is always exactly one feasible point in $\mathcal{S}(\rho)$.

Lemma 12 *For problems (D.1) with a feasible set* $A_d = \left\{ r : \sum_{i=1}^{N} r_i = K,\ r \in \mathbb{Z}_+^N \right\}$, *the selection set* $\mathcal{S}(\rho)$ *constructed above includes one and only one feasible point. Moreover, this point is the argument of* $\min_{r \in \mathcal{N}(\rho)} \|\rho - r\|$.

Proof. Since $\rho \in A_c$, it follows from (D.7) that there exist $\{\alpha_i\}_{i=1}^{M}$ that satisfy

$$\rho_j = \sum_{i=1}^{M} \alpha_i r_j^i, \quad r^i \in \mathcal{N}(\rho), \quad \sum_{i=1}^{M} \alpha_i = 1, \quad \alpha_i \geq 0 \text{ for } i = 1, ..., M$$

where $M = |\mathcal{N}(\rho)|$. Therefore,

$$\sum_{j=1}^{N} \rho_j = \sum_{j=1}^{N} \sum_{i=1}^{M} \alpha_i r_j^i = \sum_{i=1}^{M} \alpha_i \sum_{j=1}^{N} r_j^i = \sum_{i=1}^{M} \alpha_i K = K$$

Then, we can write

$$\sum_{j=1}^{N} \lfloor \rho_j \rfloor \leq K \leq \sum_{j=1}^{N} \lceil \rho_j \rceil$$

where the equality only holds for integer allocations. Since we agreed that ρ does not have integer components,

$$\sum_{j=1}^{N} \lfloor \rho_j \rfloor < K < \sum_{j=1}^{N} \lceil \rho_j \rceil$$

Note that

$$\sum_{j=1}^{N} (\lceil \rho_j \rceil - \lfloor \rho_j \rfloor) = N \tag{D.20}$$

and define the residual resource capacity m, with $0 < m < N$, as

$$m = K - \sum_{j=1}^{N} \lfloor \rho_j \rfloor \tag{D.21}$$

Also note that from (D.12) and (D.16)

$$r^0 = \lfloor \rho \rfloor + \tilde{r}^0 = \lfloor \rho \rfloor \in \mathcal{S}(\rho)$$

and from (D.8)

$$r^{o_1} = \lfloor \rho \rfloor + \tilde{r}^{o_1} = \lfloor \rho \rfloor + \sum_{k=1}^{N} e_{o_k} = \lceil \rho \rceil \in \mathcal{S}(\rho) \tag{D.22}$$

Now, observe that during the construction of $\mathcal{S}(\rho)$,

$$r^{o_l} - r^{o_{l+1}} = e_{o_l}$$

therefore,

$$\sum_{i=1}^{N} r_i^{o_l} - \sum_{i=1}^{N} r_i^{o_{l+1}} = \sum_{i=1}^{N} (e_{o_l})_i = 1 \tag{D.23}$$

Using (D.23),

$$\sum_{i=1}^{N} r^{o_1} - \sum_{i=1}^{N} r^{o_{N+1-m}} = N - m$$

and it follows from (D.22) that

$$\sum_{i=1}^{N} r^{o_{N+1-m}} = \sum_{i=1}^{N} \lceil \rho_i \rceil - N + m = K$$

where we have used (D.20) and (D.21). Therefore, $r^{o_{N+1-m}} \in \mathcal{N}(\rho)$ and the first part of the proof is complete.

By construction of the selection set $\mathcal{S}(\rho)$, the elements r^{o_l} satisfy the following

$$r_i^{o_l} = \lceil \rho_i \rceil \text{ and } r_j^{o_l} = \lfloor \rho_j \rfloor \Rightarrow \tilde{\rho}_i \geq \tilde{\rho}_j \tag{D.24}$$

Therefore, we claim that $r^{o_{N+1-m}}$ is the solution of the minimization problem

$$\min_{r \in \mathcal{N}(\rho)} \|\rho - r\| = \sqrt{\sum_{i=1}^{N} (\rho_i - r_i)^2}$$

for $\rho \in A_c$. All elements $r \in \mathcal{N}(\rho)$ can be characterized in terms of a set $\mathcal{M}_r$ of indices defined as

$$\mathcal{M}_r = \{i | r_i = \lceil \rho_i \rceil\}$$

where $|\mathcal{M}_r| = m$ for $r \in \mathcal{N}(\rho)$. One can then write an equivalent minimization problem as

$$\begin{aligned}
\min_{r\in\mathcal{N}(\rho)} \sum_{i=1}^{N}(\rho_i - r_i)^2 &= \min_{r\in\mathcal{N}(\rho)} \left[\sum_{\substack{i=1\\ r_i=\lfloor\rho_i\rfloor}}^{N} (\rho_i - r_i)^2 + \sum_{\substack{i=1\\ r_i=\lceil\rho_i\rceil}}^{N} (\rho_i - r_i)^2 \right] \\
&= \min_{r\in\mathcal{N}(\rho)} \left[\sum_{i\in I\setminus\mathcal{M}_r} \tilde{\rho}_i{}^2 + \sum_{i\in\mathcal{M}_r} (1-\tilde{\rho}_i)^2 \right]
\end{aligned}$$

For $r, r^{o_{N+1-m}} \in \mathcal{N}(\rho)$, the sets $\mathcal{M}_r$ and $\mathcal{M}_{r^{o_{N+1-m}}}$ are formed by m elements from the set $\{1, ..., N\}$. Starting at $\mathcal{M}_{r^{o_{N+1-m}}}$, one can reach any $\mathcal{M}_r$ by a series of iterations, each iteration involving the removal of one element from the set $\mathcal{M}_{r^{o_{N+1-m}}}\setminus\mathcal{M}_r$, and the addition of an element from the set $\mathcal{M}_r\setminus\mathcal{M}_{r^{o_{N+1-m}}}$. If we remove i and add j while $\tilde{\rho}_i \geq \tilde{\rho}_j$ we increase the objective value by

$$\tilde{\rho}_i^2 - \tilde{\rho}_j^2 + (1-\tilde{\rho}_j)^2 - (1-\tilde{\rho}_i)^2 = 2(\tilde{\rho}_i - \tilde{\rho}_j) \geq 0$$

Since $\mathcal{M}_{r^{o_{N+1-m}}}$ has the arguments for the m highest $\tilde{\rho}_i$, we cannot decrease the distance by moving to another $r \in \mathcal{N}(\rho)$ therefore $r^{o_{N+1-m}}$ minimizes the distance from ρ. ■

Example (Cont.): For the resource allocation problem of $K = 10$ resources over $N = 3$ users, given $\rho = [3.9, 3.9, 2.2]$, we found a selection set

$$\mathcal{S}(\rho) = \{[3,3,2], [3,4,2], [4,4,2], [4,4,3]\}$$

Observe that $[4,4,2]$ is the only element of $\mathcal{S}(\rho)$ above which is feasible, consistent with Lemma 12. In addition, note that $[4,4,2]$ is the obvious solution of the minimization problem

$$\min_{r\in\mathcal{N}(\rho)} \|\rho - r\|$$

where $\mathcal{N}(\rho) = \{[3,4,3], [4,4,2], [4,3,3]\}$.

D.3 Construction of surrogate cost functions and their gradients

Since our approach is based on iterating over the continuous state $\rho \in A_c$, yet drive the iteration process with information involving $L_d(r)$ obtained from a sample path under r, we must establish a relationship between $L_d(r)$ and $\bar{L}_c(\rho)$. The choice of $\bar{L}_c(\rho)$ is rather flexible and may depend on information pertaining to a specific model and the nature of the given cost $L_d(r)$.

Before defining $\bar{L}_c(\rho)$, we shall concentrate on surrogate cost functions $L_c(\rho, \mathcal{S}(\rho), \omega)$ (which clearly depend on a selection set and a sample path) that satisfy the following two conditions:

(C1): *Consistency*: $L_c(r, \mathcal{S}(r), \omega) = L_d(r, \omega)$ for all $r \in \mathbb{Z}_+^N$.

(C2): *Piecewise Linearity:* $L_c(\rho, \mathcal{S}(\rho), \omega)$ is a linear function of ρ over $conv(\mathcal{S}(\rho))$.

In the sequel, the '$\mathcal{S}(\rho)$' term will be dropped along with 'ω' for simplicity.

Consistency is an obvious requirement for $L_c(\rho)$. Piecewise linearity is chosen for convenience, since manipulating linear functions over $conv(\mathcal{S}(\rho))$ simplifies analysis, as will become clear in what follows.

Given some state $\rho \in A_c$ and cost functions $L_d(r^j)$ for all $r^j \in \mathcal{S}(\rho)$, it follows from **(C2)** and (D.17) that we can write

$$\rho = \sum_{j=0}^{N} \alpha_j r^j \Rightarrow L_c(\rho) = \sum_{j=0}^{N} \alpha_j L_c(r^j)$$

with $\sum_{j=0}^{N} \alpha_j = 1$, $\alpha_j \geq 0$ for all $j = 0, .., N$. Moreover, by **(C1)**, we have

$$L_c(\rho) = \sum_{j=0}^{N} \alpha_j L_d(r^j) \tag{D.25}$$

that is, $L_c(\rho)$ is a convex combination of the costs of discrete neighbors in $\mathcal{S}(\rho)$. Note that although $\mathcal{S}(\rho)$ is not unique, given $\mathcal{S}(\rho)$, the values of α_i for $i = 0, ..., N$ are unique; therefore, $L_c(\rho)$ is well defined.

We now define a surrogate cost function $\bar{L}_c(\rho)$ and the corresponding selection set $\mathcal{S}^*(\rho)$ as

$$\bar{L}_c(\rho) = \min_{\mathcal{S}(\rho)} L_c(\rho) \tag{D.26}$$

and

$$\mathcal{S}^*(\rho) = \arg\min_{\mathcal{S}(\rho)} L_c(\rho) \tag{D.27}$$

where the minimization is over all possible selection sets for the point ρ. The function $\bar{L}_c(\rho)$ satisfies the consistency condition **(C1),** but it may not be a continuous function due to changes in the selection set $\mathcal{S}^*(\cdot)$ for neighboring points.

Next, if we are to successfully use the iterative scheme described by (D.4)-(D.5), we need information of the form $H_n(\rho_n, r_n, \omega_n)$ following the nth step of the on-line optimization process. Typically, this information is contained in an estimate of the gradient. Our next objective, therefore, is to seek the selection-set-dependent sample gradient $\nabla L_c(\rho)$ expressed in terms of directly observable sample path data.

D.3.1 Gradient evaluation

The gradient information necessary to drive the stochastic approximation part of the surrogate method is evaluated depending on the form of the cost function. Gradient estimation for separable cost functions is significantly simpler and is discussed in the next section.

Since $L_c(\rho)$ is a linear function on the convex hull defined by $\mathcal{S}(\rho)$, one can write

$$L_c(\rho) = \sum_{i=1}^{N} \beta_i \rho_i + \beta_0 \tag{D.28}$$

where

$$\beta_i = \frac{\partial L_c(\rho)}{\partial \rho_i},\ i = 1, ..., N$$

and $\beta_0 \in \mathbb{R}$ is a constant. Note that the β_i values depend on the selection set $\mathcal{S}(\rho)$, which, as already pointed out, may not be unique.

For any $r^j \in \mathcal{S}(\rho)$, one can use (D.28) and **(C1)** to write

$$L_d(r^j) = \sum_{i=1}^{N} \beta_i r_i^j + \beta_0$$

Note that the values $L_d(r^j)$ are either observable or can be evaluated using Concurrent Estimation or Perturbation Analysis techniques (see [41], [20]) despite the fact that $r^j \in \mathcal{S}(\rho)$ may be infeasible, i.e., *having infeasible points in the selection set does not affect our ability to obtain gradients.* One can now rewrite the equation above as

$$\begin{bmatrix} e & \mathbf{R} \end{bmatrix} \beta = L$$

where e is an $(N+1)-$dimensional column vector with all entries equal to 1, $\beta = [\beta_0, ..., \beta_N]'$, $\mathbf{R}$ is the matrix whose rows are $r^j \in \mathcal{S}(\rho)$, and L is the column vector of costs for these discrete states. Since we have constructed $\mathcal{S}(\rho)$ so that $\begin{bmatrix} e & \mathbf{R} \end{bmatrix}$ is non-singular, the gradient given by $\nabla L_c(\rho) = [\beta_1, ..., \beta_N]'$, can be obtained from the last equation as

$$\nabla L_c(\rho) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} e & \mathbf{R} \end{bmatrix}^{-1} L \tag{D.29}$$

where $\mathbf{I}$ is the identity matrix of dimension N and $\mathbf{0}$ is the N-dimensional vector of zeros. Substituting from equation (D.18), the gradient can be written as

$$\nabla L_c(\rho) = \begin{bmatrix} -\hat{e}_{N+1-\bar{o}_1} + \hat{e}_{N+2-\bar{o}_1} \\ \vdots \\ -\hat{e}_{N+1-\bar{o}_N} + \hat{e}_{N+2-\bar{o}_N} \end{bmatrix} \begin{bmatrix} L_d(r^0) \\ L_d(r^{o_N}) \\ \vdots \\ L_d(r^{o_1}) \end{bmatrix} \tag{D.30}$$

Therefore,

$$\nabla_j L_c(\rho) = L_d(r^j) - L_d(r^k) \tag{D.31}$$

where k satisfies $\bar{o}_j + 1 = \bar{o}_k$, i.e., $r^j - r^k = e_j$. As pointed out in the Introduction, this is a significant simplification over the gradient evaluation used in our earlier work [33]. Moreover, this analysis allows us to combine the algorithm for determining the selection set given in the previous section with the gradient estimation component of our approach to obtain the following:

- Initialize the index set $I = \{1, ..., N\}$
- $r^i = \lceil \rho \rceil$ where $i = \arg\min_{j \in I} \tilde{\rho}_j$
- $OC = L_d(r^i)$
- $oi = i$
- $I = I \backslash \{i\}$
- While $I \neq \emptyset$:

1. $r^k = r^{oi} - e_{oi}$ where $k = \arg\min_{j \in I} \tilde{\rho}_j$
2. $\nabla_{oi} L_c(\rho) = OC - L_d(r^k)$
3. $OC = L_d(r^k)$
4. $oi = k$
5. $I = I \backslash \{k\}$

- $r^0 = \lfloor \rho \rfloor$
- $\nabla_{oi} L_c(\rho) = OC - L_d(r^0)$

Example (Cont.): For the example we have been using throughout the paper, consider the cost function

$$J(r) = \|r - [2, 5, 3]\|^2$$

Let $\rho_n = [3.9, 3.9, 2.2]$, for which we found $\mathcal{S}(\rho) = \{[3, 3, 2], [3, 4, 2], [4, 4, 2], [4, 4, 3]\}$ with $r^1 = [4, 4, 2]$, $r^2 = [3, 4, 2]$, $r^3 = [4, 4, 3]$, and $r^0 = [3, 3, 2]$. The gradient at this point can, therefore, be evaluated using (D.31) to give

$$\nabla L_c(\rho_n) = \left[\begin{array}{c} J_d([4,4,2]) - J_d([3,4,2]) \\ J_d([3,4,2]) - J_d([3,3,2]) \\ J_d([4,4,3]) - J_d([4,4,2]) \end{array} \right] = \left[\begin{array}{c} 3 \\ -3 \\ -1 \end{array} \right]$$

Using $\eta_n = 0.5$ in (D.4):

$$\begin{aligned} \rho_{n+1} &= \pi_{n+1}[\rho_n - \eta_n \nabla L_c(\rho_n)] \\ &= \pi_{n+1}[[2.4, 5.4, 2.7]] \\ &= [2.2, 5.2, 2.6] \end{aligned}$$

where we have used the projection π to map the point $[2.4, 5.4, 2.7]$ into a feasible point $[2.2, 5.2, 2.6] \in A_c$. For this example, π can be defined as follows:

$$\pi[\bar{\rho}] = \arg \min_{\sum_{i=1}^{N} \rho_i = 10} \|\rho - \bar{\rho}\|$$

D.3.2 Projection Mapping

The projection mapping π is a crucial element of our method and has a very significant effect on the convergence. In this section, we discuss a projection mapping which can be used for resource allocation problems with feasible sets

$$A_d = \left\{ r : \sum_{i=1}^{N} r_i = K,\ r \in \mathbb{Z}_+^N \right\}$$

Consider the optimization problem

$$\min_{\rho} \sum_{i=1}^{N} (\rho_i - \bar{\rho}_i)^2$$

subject to

$$\begin{aligned} \rho_i &\geq 0 \\ \sum_{i=1}^{N} \rho_i &= K \end{aligned}$$

The solution to this optimization problem, which we will denote by $\pi(\bar{\rho})$, is the closest point in the feasible set A_c to the point $\bar{\rho}$. Note that a π projection to a closed convex set defined in this manner is continuous and nonexpansive, therefore it guarantees convergence (see [33]).

Let us consider the relaxed problem

$$\min_{\rho_i \geq 0} \sum_{i=1}^{N} \left[(\rho_i - \bar{\rho}_i)^2 - \lambda \rho_i \right] + \lambda K$$

The necessary optimality conditions are

$$\begin{aligned} [2(\rho_i - \bar{\rho}_i) - \lambda] &= 0 \text{ for } \rho_i > 0 \\ [2(\rho_i - \bar{\rho}_i) - \lambda] &> 0 \text{ for } \rho_i = 0 \\ \sum_{i=1}^{N} \rho_i &= K \end{aligned}$$

or equivalently

$$\begin{aligned} \rho_i &= \bar{\rho}_i + \frac{\lambda}{2} \text{ for } \rho_i > 0 \\ \rho_i &> \bar{\rho}_i + \frac{\lambda}{2} \text{ for } \rho_i = 0 \\ \sum_{i=1}^{N} \rho_i &= K \end{aligned}$$

i.e.,

$$\begin{aligned}\rho_i &= \max(0, \bar{\rho}_i + \frac{\lambda}{2}) \\ \sum_{i=1}^{N} \rho_i &= K\end{aligned}$$

These equations suggest the following algorithm for the π projection:

Projection Algorithm:

- Initialize $\lambda^0 = \frac{2}{N}(K - \sum_{i=1}^{N} \max(0, \bar{\rho}_i))$
- If some stopping condition is not satisfied:

1. For $i = 1, 2, ...N, \qquad \rho_i = \max(0, \bar{\rho}_i + \frac{\lambda}{2})$
2. $\lambda \leftarrow \lambda + \frac{2}{N}(K - \sum_{i=1}^{N} \rho_i)$

- Set $\pi[\bar{\rho}] = \rho$.

A common stopping condition we have used in our work (see also Section D.6) is $\left|K - \sum_{i=1}^{N} \rho_i\right| \leq \varepsilon$, for some small $\varepsilon > 0$. Then, the vector ρ is rescaled

$$\rho \leftarrow \frac{K}{\sum_{i=1}^{N} \rho_i} \rho$$

to satisfy the capacity constraint. The error introduced while rescaling is small and it is a monotonically increasing function of ε. Note that there is a trade-off between the number of iterations needed and the size of the resulting error term determined by the selection of ε.

D.3.3 Separable cost functions

Suppose that the cost function, $L_d(\cdot)$, is *separable* in the sense that it is a sum of component costs each dependent on its local state only, i.e., let

$$L_d(r) = \sum_{i=1}^{N} L_{d,i}(r_i) \tag{D.32}$$

In this case, our approach is significantly simplified. In particular, from (D.31) and (D.32), we can write

$$\begin{aligned}
\nabla_j L_c(\rho) &= L_d(r^j) - L_d(r^k) \\
&= \sum_{i=1}^{N} L_{d,i}(r_i^j) - \sum_{i=1}^{N} L_{d,i}(r_i^k) \\
&= \sum_{i=1}^{N} [L_{d,i}(r_i^j) - L_{d,i}(r_i^k)] \\
&= L_{d,j}(r_j^j) - L_{d,j}(r_j^j - 1)
\end{aligned}$$

where k satisfies $\bar{o}_j + 1 = \bar{o}_k$, i.e., $r^j - r^k = e_j$. Note that $r_j^j = \lceil \rho_j \rceil$ and $r_j^j - 1 = \lfloor \rho_j \rfloor$; therefore,

$$\nabla_j L_c(\rho) = L_{d,j}(\lceil \rho_j \rceil) - L_{d,j}(\lfloor \rho_j \rfloor) \tag{D.33}$$

This result indicates that for separable cost functions estimating sensitivities does not require the determination of a selection set; we can instead simply pick a feasible neighbor (preferably the closest feasible neighbor to ρ) and apply Perturbation Analysis (PA) techniques to determine the gradient components through (D.33). There are a number of PA techniques developed precisely for evaluating the effect of decreasing and increasing the number of resources allocated to user i; for example, estimating the sensitivity of packet loss in a radio network with respect to adding/removing a transmission slot available to the ith user [19], [82]. In [34] a PA technique is used together with our methodology to solve a call admission problem (with a separable cost function) over a communication network where there are capacity constraints on each node, while there is no total capacity constraint for the network.

D.4 Recovery of optimal discrete states

Our ultimate goal remains the solution of (D.1), that is the determination of $r^* \in A_d$ that solves this optimization problem. Our approach is to solve (D.2) by iterating on $\rho \in A_c$ and, at each step, transforming ρ through some $f \in \mathcal{F}_\rho$. The connection between ρ and $r = f(\rho)$ for each step is therefore crucial, as is the relationship between ρ^* and $f(\rho^*)$ when and if this iterative process comes to an end identifying a solution ρ^* to the surrogate problem (D.2).

The following theorem identifies a key property of the selection set $\mathcal{S}^*(\rho^*)$ of an *optimal* surrogate state ρ^*.

Theorem 13 *Let ρ^* minimize $\bar{L}_c(\rho)$ over A_c. If $r^* = \arg\min_{r \in \mathcal{S}^*(\rho^*)} L_d(r) \in \mathcal{N}(\rho^*)$, i.e., the minimal cost element r^* of the selection set $\mathcal{S}^*(\rho^*)$ corresponding to $\bar{L}_c(\rho^*)$ is feasible, then r^* minimizes $L_d(r)$ over A_d and satisfies $L_d(r^*) = \bar{L}_c(\rho^*)$.*

Proof. By (D.25), the optimal surrogate state $\rho^* = \arg\min_{\rho \in A_c} \bar{L}_c(\rho)$ satisfies

$$\bar{L}_c(\rho^*) = \sum_{j=0}^{N} \alpha_j L_d(r^j)$$

where $\sum_{j=0}^{N} \alpha_j = 1$, $r^j \in \mathcal{S}^*(\rho^*)$, $\alpha_j \geq 0$ for $j = 0, .., N$. Then,

$$\bar{L}_c(\rho^*) = \sum_{j=0}^{N} \alpha_j L_d(r^j) \geq \sum_{j=0}^{N} \alpha_j L_d(r^*) = L_d(r^*) \tag{D.34}$$

regardless of the feasibility of r^*.

Next, note that $A_d \subset A_c$ and $\bar{L}_c(r) = L_d(r)$ for any $r \in A_d$. Therefore, if $r^* \in \mathcal{N}(\rho^*)$, then

$$L_d(r^*) = \bar{L}_c(r^*) \geq \bar{L}_c(\rho^*) \tag{D.35}$$

In view of (D.34) and (D.35), we then get

$$L_d(r^*) \leq \bar{L}_c(\rho^*) \leq L_d(r^*)$$

It follows that

$$L_d(r^*) = \bar{L}_c(\rho^*)$$

that is, r^* is optimal over A_d. Finally, since r^* is one of the discrete feasible neighboring states of ρ^*, i.e. $r^* \in \mathcal{N}(\rho^*)$, we have $r^* = f(\rho^*)$ for some $f \in \mathcal{F}_{\rho^*}$. ■

Corollary 14 *For unconstrained problems, let ρ^* minimize $\bar{L}_c(\rho)$. Then,*

$$r^* = \arg \min_{r \in \mathcal{S}^*(\rho^*)} L_d(r)$$

minimizes $L_d(r)$ and satisfies $L_d(r^*) = \bar{L}_c(\rho^*)$.

Proof. If problem (D.1) is unconstrained, then trivially $r^* = \arg\min_{r \in \mathcal{S}^*(\rho^*)} L_d(r) \in \mathcal{N}(\rho^*)$ and the result follows. ■

An interesting example of an unconstrained problem is that of lot sizing in manufacturing systems (see [22]) where the sizes of lots of different parts being produced may take any (non-negative) integer value. Clearly, Corollary 14 also holds for problems where the optimal point is in the interior of the feasible set where the constraints are not active, i.e., $\mathcal{N}(\rho^*) = \mathcal{C}(\rho^*)$.

If there are active constraints around the optimal point ρ^*, i.e., $\mathcal{N}(\rho^*) \neq \mathcal{C}(\rho^*)$, and there are infeasible points in the selection set $\mathcal{S}(\rho^*)$, then, if one of these infeasible points has the minimal cost, the recovery of the optimal as a feasible neighbor of ρ^* becomes difficult to guarantee theoretically, even though empirical evidence shows that this is indeed the case. This is the price to pay for the generalization of the surrogate problem method we have presented here through the introduction of a selection set that allows the inclusion of infeasible points. However, if the cost function $L_d(r)$ is "smooth" in some sense, the minimal cost element of $\mathcal{N}(\rho^*)$ will in general be either an optimal or a near-optimal point as stated in the next lemma.

Lemma 15 *If the cost function $L_d(r)$ satisfies*

$$\left|L_d(r^1) - L_d(r^2)\right| \leq c_\omega \left\|r^1 - r^2\right\|, \quad c_\omega < \infty \tag{D.36}$$

then all $r \in \mathcal{N}(\rho^*)$ satisfy

$$L_d(r) \leq \bar{L}_c(\rho^*) + c_\omega \sqrt{N} \tag{D.37}$$

Proof. Note that $\mathcal{S}^*(\rho^*) \subset \mathcal{C}(\rho^*)$ and $\mathcal{N}(\rho^*) \subset \mathcal{C}(\rho^*)$. It is easy to show that for $r^1, r^2 \in \mathcal{C}(\rho^*)$

$$\left\|r^1 - r^2\right\| \leq \sqrt{N}$$

By (D.34), there exists $r^* \in \mathcal{S}^*(\rho^*)$ that satisfies $\bar{L}_c(\rho^*) \geq L_d(r^*)$ regardless of its feasibility. For $r \in \mathcal{N}(\rho^*)$, we can write $\bar{L}_c(\rho^*) \leq L_d(r)$, therefore

$$|L_d(r) - L_d(r^*)| = L_d(r) - L_d(r^*) \geq L_d(r) - \bar{L}_c(\rho^*) \tag{D.38}$$

By assumption (D.36),

$$|L_d(r) - L_d(r^*)| \leq c_\omega \|r - r^*\| \leq c_\omega \sqrt{N} \tag{D.39}$$

Hence, from (D.38) and (D.39),

$$L_d(r) \leq \bar{L}_c(\rho^*) + c_\omega \sqrt{N}$$

■

In practice, for many cost metrics such as throughput or mean system time in queueing models, it is common to have the costs in the neighborhood of an optimal point be relatively close, in which case the value of c_ω is small and (D.37) is a useful bound.

D.5 Optimization Algorithm

Summarizing the results of the previous sections and combining them with the basic scheme described by (D.4)-(D.5), we obtain the following optimization algorithm for the solution of the basic problem in (D.1):

- Initialize $\rho_0 = r_0$ and perturb ρ_0 to have all components non-integer.
- For any iteration $n = 0, 1, \ldots$:
  1. Determine $\mathcal{S}(\rho_n)$ [using the construction of Theorem 11; recall that this set is generally not unique].
  2. Select $f_n \in \mathcal{F}_{\rho_n}$ such that $r_n = \arg\min_{r \in \mathcal{N}(\rho_n)} \|r - \rho_n\| = f_n(\rho_n) \in \mathcal{N}(\rho_n)$.
  3. Operate at r_n to collect $L_d(r^i)$ for all $r^i \in \mathcal{S}(\rho_n)$ [using Concurrent Estimation or some form of Perturbation Analysis; or, if feasible, through off-line simulation].
  4. Evaluate $\nabla L_c(\rho_n)$ [using (D.31)].

5. Update the continuous state: $\rho_{n+1} = \pi_{n+1}[\rho_n - \eta_n \nabla L_c(\rho_n)]$.
6. If some stopping condition is not satisfied, repeat steps for $n+1$. Else, set $\rho^* = \rho_{n+1}$.

- Obtain the optimal (or the near optimal) state as one of the neighboring feasible states in the set $\mathcal{N}(\rho^*)$.

Note that for separable cost functions, steps 1-6 can be replaced by

1. Select f_n such that $r_n = \arg\min_{r\in\mathcal{N}(\rho_n)} \|r - \rho_n\| = f_n(\rho_n) \in \mathcal{N}(\rho_n)$.
2. Operate at r_n to evaluate $\nabla L_c(\rho_n)$ using Perturbation Analysis and (D.33).
3. Update the continuous state: $\rho_{n+1} = \pi_{n+1}[\rho_n - \eta_n \nabla L_c(\rho_n)]$.
4. If some stopping condition is not satisfied, repeat steps for $n+1$. Else, set $\rho^* = \rho_{n+1}$.

The surrogate part of this algorithm is a stochastic approximation scheme with projection whose convergence was analyzed in [33] and references therein.

Note that ideally we would like to have $\nabla J_c(\rho_n)$ be the cost sensitivity driving the algorithm. Since this information is not always available in a stochastic environment and since $J_c(\rho_n) = E[\bar{L}_c(\rho_n,\omega)]$, the stochastic approximation algorithm uses $\nabla \bar{L}_c(\rho_n,\omega)$ as an estimate and under some standard assumptions on the estimation error ε_n where

$$\nabla J_c(\rho_n) = \nabla \bar{L}_c(\rho_n,\omega) + \varepsilon_n$$

the convergence is guaranteed. In order to get $\nabla \bar{L}_c(\rho_n,\omega)$, however, one needs to consider all possible selection sets. In this algorithm we utilize only one of those selection sets and approximate $\nabla \bar{L}_c(\rho_n,\omega)$ with $\nabla L_c(\rho_n, S(\rho_n),\omega)$. This approximation introduces yet another error term $\bar{\varepsilon}_n$ where

$$\nabla \bar{L}_c(\rho_n,\omega) = \nabla L_c(\rho_n, S(\rho_n),\omega) + \bar{\varepsilon}_n$$

Note that this error term $\bar{\varepsilon}_n$ exists regardless of stochasticity, unless the cost function $L_d(.)$ is separable (all selection sets will yield the same sensitivity for separable cost functions). We can combine error terms to define $\epsilon_n = \bar{\varepsilon}_n + \varepsilon_n$ and write

$$\nabla J_c(\rho_n) = \nabla L_c(\rho_n, S(\rho_n),\omega) + \epsilon_n$$

If the augmented error term ϵ_n satisfies the standard assumptions, then convergence of the algorithm to the optimal follows in the same way as presented in [33].

D.6 Numerical Examples and Applications

We first illustrate our approach by means of a simple deterministic example, followed by a more challenging stochastic optimization application for a classical problem in manufacturing systems.

Example 1: Consider an allocation problem of $K = 20$ resources over $N = 4$ users so as to minimize the convex cost function $J_d(r)$ defined as

$$J_d(r) = \|r - [4, 5, 3, 8]\|^2$$

Suppose the initial state is $\rho_0 = [1.8, 9.1, 6.2, 2.9]$. Note that the set of feasible neighboring states $\mathcal{N}(\rho_0)$ is

$$\mathcal{N}(\rho_0) = \{[2, 10, 6, 2], [2, 9, 7, 2], [2, 9, 6, 3], [1, 10, 7, 2], [1, 10, 6, 3], [1, 9, 7, 3]\}$$

Following the steps shown in the algorithm of Section D.5, we have:

1. Determine the selection set $\mathcal{S}(\rho_0)$

$$\mathcal{S}(\rho_0) = \{[1, 9, 6, 2], [1, 9, 6, 3], [2, 9, 6, 3], [2, 9, 7, 3], [2, 10, 7, 3]\}$$

2. Select $r_0 = f_0(\rho_0) \in \mathcal{N}(\rho_0)$:

$$r_0 = [2, 9, 6, 3]$$

3. Evaluate cost functions for states in $\mathcal{S}(\rho_0)$:

$$\begin{aligned} J_d([1,9,6,2]) &= 70 \qquad J_d([1,9,6,3]) = 59 \qquad J_d([2,9,6,3]) = 54 \\ J_d([2,9,7,3]) &= 61 \qquad J_d([2,10,7,3]) = 70 \end{aligned}$$

4. Evaluate the gradient of the cost at ρ_0

$$\begin{aligned} (\nabla J_c(\rho_0))_1 &= J_d([2,9,6,3]) - J_d([1,9,6,3]) = -5 \\ (\nabla J_c(\rho_0))_2 &= J_d([2,10,7,3]) - J_d([2,9,7,3]) = 9 \\ (\nabla J_c(\rho_0))_3 &= J_d([2,9,7,3]) - J_d([2,9,6,3]) = 7 \\ (\nabla J_c(\rho_0))_4 &= J_d([1,9,6,3]) - J_d([1,9,6,2]) = -11 \end{aligned}$$

Therefore,

$$\nabla J_c(\rho_0) = \begin{bmatrix} -5 \\ 9 \\ 7 \\ -11 \end{bmatrix}$$

5. Update the surrogate state:

$$\rho_1 = \pi_1[\rho_0 - \eta_0 \nabla J_c(\rho_0)]$$

6. If the stopping condition is not satisfied, go to step 1 and repeat with ρ_{n+1} replacing ρ_n for $n = 0, 1, ...$

Using a step size sequence $\eta_n = 0.5/(n+1)$, the following table shows the evolution of the algorithm for the first few steps. Note that the optimal allocation $[4, 5, 3, 8]$ is reached after a single step.

STEP	ρ	r	$J_c(\rho)$	$J(r)$
0	$[1.800, 9.100, 6.200, 2.900]$	$[2, 9, 6, 3]$	56.84	54
1	$[4.300, 4.600, 2.700, 8.400]$	$[4, 5, 3, 8]$	0.50	0
2	$[4.050, 4.850, 2.950, 8.150]$	$[4, 5, 3, 8]$	0.05	0

Table D.1: Optimal Resource Allocation

Example 2: Consider a manufacturing system formed by five stages in series. The arrival process to the system is Poisson with rate $\lambda = 1.0$ and the service processes are all exponential with rates $\mu_1 = 2.0$, $\mu_2 = 1.5$, $\mu_3 = 1.3$, $\mu_4 = 1.2$, and $\mu_5 = 1.1$. Note that Poisson arrival process and exponential service times are not required by the algorithm. They are chosen for simplicity of the simulations.

We would like to allocate kanban (tickets) to stages $2-5$, to minimize a cost function that has two components

$$J(r) = J_1(r) + J_2(r)$$

where $r \in \mathbb{Z}_+^4$ is the vector of kanban allocated to stages $2-5$. The first component $J_1(r)$ is the average system time for jobs and the second component $J_2(r)$ is a cost on the number of kanban allocated defined as

$$J_2(r) = c\left|K - \sum_{i=1}^{4} r_i\right|$$

For large enough c, the second component $J_2(r)$ dominates the cost; therefore, a capacity constraint of the form

$$\sum_{i=1}^{4} r_i = K$$

is enforced. The problem, then, can be written as

$$\min_{\sum_{i=1}^{4} r_i = K} J_1(r)$$

which was considered in [67] with $K = 13$. The surrogate method for the same problem performs as follows:

At each iteration we observe 100 departures and use the decreasing step size $\eta_n = \frac{140}{n}$. The optimal allocation is observed as $[1, 3, 4, 5]$ which matches the result from [67]. It is worthwhile noting that this optimal point is identified within 13 iterations, illustrating the convergence speed of this method.

Iterations	r	$J(r)$
1	$[3,3,3,4]$	0.798133
2	$[1,2,2,8]$	0.781896
3	$[1,5,4,8]$	0.767171
4	$[1,4,6,7]$	0.746568
5	$[1,4,6,7]$	0.761161
6	$[1,4,6,6]$	0.709394
7	$[1,3,5,6]$	0.827928
8	$[1,3,5,6]$	0.788815
9	$[1,3,5,5]$	0.730709
10	$[1,3,5,6]$	0.742748
11	$[1,3,5,5]$	0.791522
12	$[1,2,5,5]$	0.865436
13	$[1,3,4,5]$	0.795680
14	$[1,3,4,5]$	0.738700
15	$[1,3,4,5]$	0.857133
16	$[1,3,4,5]$	0.679464
17	$[1,3,4,5]$	0.875472
18	$[1,3,4,5]$	0.840447

Table D.2: Optimal Kanban Allocation

D.6.1 Multicommodity Resource Allocation Problems

An interesting class of discrete optimization problems arises when Q different types of resources must be allocated to N users. The corresponding optimization problem we would like to solve is

$$\min_{r \in A_d} J(r)$$

where $r = [r_{1,1}, \ldots, r_{1,Q}, \cdots, r_{N,1}, \ldots, r_{N,Q}]$ is the allocation vector and $r_{i,q}$ is the number of resources of type q allocated to user i. A typical feasible set A_d is defined by the capacity constraints

$$\sum_{i=1}^{N} r_{i,q} \leq K_q, \quad q = 1, \ldots, Q$$

and possibly additional constraints such as $\beta_i \leq r_{i,q} \leq \gamma_i$ for $i = 1, \ldots, N$. Aside from the fact that such problems are of higher dimensionality because of the Q different resource types that must be allocated to each user, it is also common that they exhibit multiple local minima. Examples of such problems are encountered in operations planning that involve N tasks to be simultaneously performed, each task i requiring a "package" of resources $(r_{i,1}, \ldots, r_{i,Q})$ in order to be carried out. The natural trade-off involved is between carrying out fewer tasks each with a high probability of success (because each task is provided adequate resources) and carrying out more tasks each with lower probability of success.

The "surrogate problem" method provides an attractive means of dealing with these problems with local minima because of its convergence speed. Our approach for solving these problems is to randomize over the initial states r_0 (equivalently, ρ_0) and seek a (possibly local) minimum corresponding to this initial point. The process is repeated for different, randomly selected, initial states so as to seek better solutions. For deterministic problems, the best allocation seen so far is reported as the optimal. For stochastic problems, we adopt the stochastic comparison approach in [36]. The algorithm is run from a randomly selected initial point and the cost of the corresponding final point is compared with the cost of the "best point seen so far". The stochastic comparison test in [36] is applied to determine the "best point seen so far" for the next run. Therefore, the surrogate problem method can be seen as a complementary component for random search algorithms that exploits the problem structure to yield better generating probabilities (as discussed in [36]), which will eliminate (or decrease) the visits to poor allocations enabling them to be applied on-line.

In what follows we consider a problem with $N = 16$, $Q = 2$, and $K_1 = 20$, $K_2 = 8$. We then seek a 32−dimensional vector $r = [r_{1,1}, r_{1,2}, \cdots, r_{16,1}, \ldots, r_{16,2}]$ to maximize a reward function of the form

$$J(r) = \sum_{i=1}^{16} J_i(r) \tag{D.40}$$

subject to

$$\sum_{i=1}^{N} r_{i,1} \leq 20, \qquad \sum_{i=1}^{N} r_{i,2} \leq 8$$

The reward functions $J_i(r)$ we will use in this problem are defined as

$$J_i = V_i P_i^0(r) - C_1 r_{i,1} P_i^1(r) - C_2 r_{i,2} P_i^2(r) \tag{D.41}$$

In (D.41), V_i represents the "value" of successfully completing the ith task and $P_i^0(r)$ is the probability of successful completion of the ith task under an allocation r. In addition, C_q is the cost of a resource of type q, where $q = 1, 2$, and $P_i^q(r)$ is the probability that a resource of type q is completely consumed or lost during the execution of the ith task under an allocation r. A representative example of a reward function for a single task with $V_i = 150$ is shown in Fig. D.1.

The cost values of resource types are $C_1 = 20$ and $C_2 = 40$, and the values for tasks we will use in this problem range between 50 and 150.

The surrogate method is executed from random initial points and the results for some runs are shown in Fig. D.2. Note that due to local maxima, some runs yield suboptimal results. However, in all cases convergence is attained extremely fast, enabling us to repeat the optimization process multiple times with different initial points in search of the global maximum. Although it is infeasible to identify the actual global maximum, we have compared our approach to a few heuristic techniques and pure random search methods and found the "surrogate problem" method to outperform them.

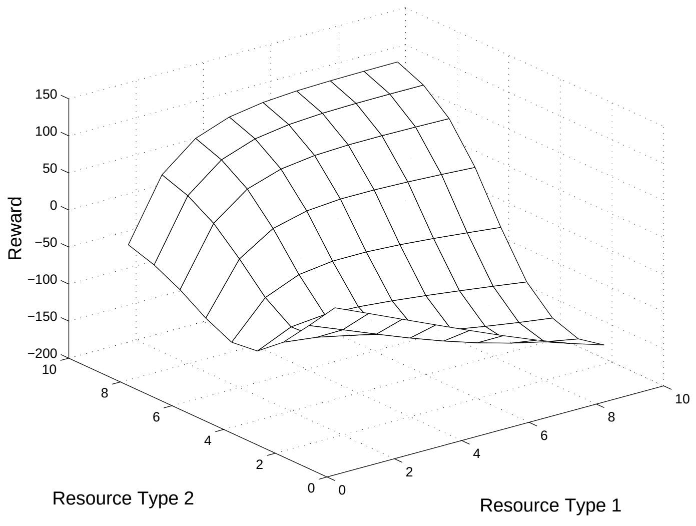


Figure D.1: A typical reward function $J_i(r_{i,1}, r_{i,2})$

D.7 Conclusions

In this paper we have generalized the methodology presented in [33] for solving stochastic discrete optimization problems. In particular, we have introduced the concept of a "selection set" associated with every surrogate state $\rho \in A_c$ and modified the definition of the surrogate cost function $\bar{L}_c(\rho)$ so that the method can be applied to arbitrary constraint sets and is computationally more efficient.

As in [33], the discrete optimization problem was transformed into a "surrogate" continuous optimization problem which was solved using gradient-based techniques. It was shown that, under certain conditions, the solution of the original problem is recovered from the optimal surrogate state. A key contribution of the methodology is its *on-line* control nature, based on the actual data from the underlying system. One can therefore see that this approach is intended to combine the advantages of a stochastic approximation type of algorithm with the ability to obtain sensitivity estimates with respect to discrete decision variables. This combination leads to very fast convergence to the optimal point.

Using this approach, we have also tackled a class of particularly hard multicommodity discrete optimization problems, where multiple local optima typically exist. Exploiting the convergence speed of the surrogate method, we presented a procedure where the algorithm is started from multiple random initial states in an effort to determine the global optimum.

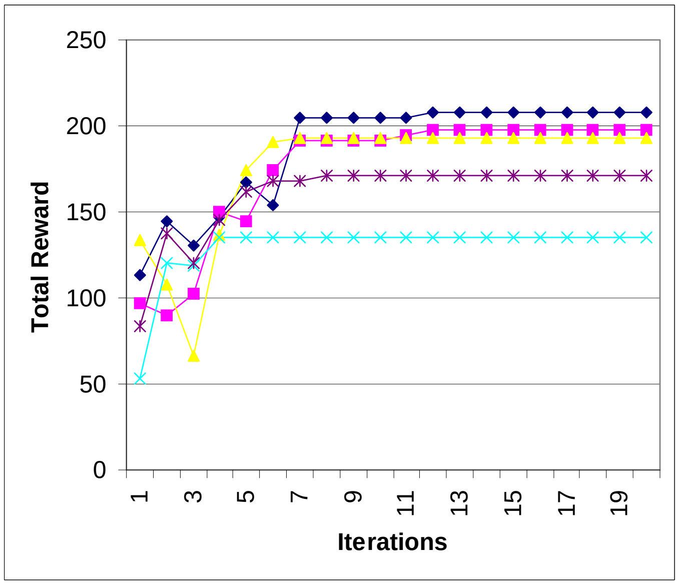


Figure D.2: Algorithm convergence under different initial points

Bibliography

[1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, Wiley, New York, NY, 1989.

[2] S. Agrawal, *Metamodeling*, MIT Press, 1985.

[3] D. Anderson, T. Frivold, and A. Valdes, *Next-generation intrusion detection expert system (NIDES): A summary.* Computer Science Laboratory, SRI-CSL-95-07, May 1995, 1995.

[4] D. Anick, D. Mitra, and M. Sondhi, *Stochastic theory of a data-handling system with multiple sources*, The Bell System Technical Journal, 61 (1982), pp. 1871–1894.

[5] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, *A framework for Internet traffic engineering (Internet Draft)*, tech. report, IETF, May 2000.

[6] C. M. Barnhart, J. E. Wieselthier, and A. Ephremides, *Admission control policies for multihop wireless networks*, Wireless Networks, 1 (1995), pp. 373–387.

[7] E. B. Baum and D. Haussler, *What size net gives valid generalization*, Neural Computation, 1 (1989), pp. 151–160.

[8] H. Bruneel and I. Wuyts, *Analysis of discrete-time multiserver queueing models with constant service times*, Operations Research Letters **15**, (1994), pp. 231–236.

[9] J. Cao, W. Cleveland, D. Lin, and D. Sun, *The effect of statistical multiplexing on Internet packet traffic: theory and empirical study*, tech. report, Bell Labs, 2001.

[10] G. Carpenter and S. Grossberg, *ART 2: Self-organization of stable category recognition codes for analog input patterns*, Applied Optics, (1987).

[11] C. Cassandras, *Discrete Event Systems, Modeling and Performance Analysis*, Irwin, 1993.

[12] C. Cassandras and W.-B. Gong, *Enabling technologies for real-time simulation*, tech. report, Univ. of Mass., Dec. 1996.

[13] ——, *Real-time simulation technologies for complex systems*, tech. report, Boston University, Nov. 1999.

[14] C. Cassandras, W.-B. Gong, C. Liu, C. Panayiotou, and D. Pepyne, *Simulation-driven metamodeling of complex systems using neural networks*, in Proceedings of 19th SPIE Conference, Apr. 1998.

[15] C. Cassandras, G. Sun, and C. Panayiotou, *Stochastic fluid models for control and optimization of systems with quality of service requirements*, in Proceedings of the 2001 IEEE Conference on Decision and Control, 2001, pp. 1917–1922.

[16] C. Cassandras, G. Sun, C. Panayiotou, and Y. Wardi, *Perturbation analysis and control of two-class stochastic fluid models for communication networks*, IEEE Transactions on Automatic Control, (2002). Submitted.

[17] C. Cassandras, Y. Wardi, B. Melamed, G. Sun, and C. Panayiotou, *On-line gradient estimation for control and optimization of stochastic fluid models*, IEEE Transactions on Automatic Control, (2001). To Appear.

[18] C. G. Cassandras, L. Dai, and C. G. Panayiotou, *Ordinal optimization for deterministic and stochastic resource allocation.*, IEEE Trans. Automatic Control, 43 (1998), pp. 881–900.

[19] C. G. Cassandras and V. Julka, *Scheduling policies using marked/phantom slot algorithms*, Queueing Systems: Theory and Applications, 20 (1995), pp. 207–254.

[20] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.

[21] C. G. Cassandras and C. G. Panayiotou, *Concurrent sample path analysis of discrete event systems*, Journal of Discrete Event Dynamic Systems: Theory and Applications, 9 (1999), pp. 171–195.

[22] C. G. Cassandras and R. Yu, *A 'surrogate problem' approach for lot size optimization in manufacturing systems*, Proc. of 2000 American Control Conference, (2000), pp. 3279–3283.

[23] C.Cassandras, W.-B.Gong, C.Liu, C.Panayiotou, and D.Pepyne, *Simulation-driven metamodeling of complex systems using neural networks*, in Proceedings of 19th SPIE Conference, APR 1998.

[24] R. C. H. Cheng, *Regression metamodeling in simulation using Bayesian methods*, in Proceedings of Winter Simulation Conference, Dec 1999, pp. 330–335.

[25] R. Cruz, *A calculus for network delay, Part 1: Network elements in isolation*, IEEE Transactions on Information Theory, (1991).

[26] S. Fahlman and C. Lebiere, *The cascade-correlation learning architecture*, tech. report, Carnegie Mellon Univ. CMU-CS-90-100, Feb. 1990.

[27] L. Fausett, *Fundamentals of Neural Networks: Architecture, Algorithms and Applications*, Prentice Hall, 1994.

[28] A. Feldmann, A. Gilbeert, and W. Williger, *Data networks as cascades: investigating the multifractal nature of Internet WAN traffic*, in Proc. of the ACM/Sigcomm'98, September 1998, pp. 42–55.

[29] A. Frantz, F.K.and Ellor, *Model abstraction techniques*, tech. report, Computer Sciences Corp. CDRL A003, Jan. 1995.

[30] L. Friedman, *The Simulation Metamodel*, Kluwer, 1996.

[31] P. Glasserman, *Gradient Estimation via Perturbation Analysis*, Kluwer Academic Pub., 1991.

[32] K. Gokbayrak and C. G. Cassandras, *A generalized 'surrogate problem' methodology for on-line stochastic discrete optimization*, J. of Optimization Theory and Applications. Submitted 2001.

[33] K. Gokbayrak and C. G. Cassandras, *Stochastic discrete optimization using a surrogate problem methodology*, in Proceedings of 20th SPIE Conference, Orlando, Apr. 1999.

[34] K. Gokbayrak and C. G. Cassandras, *Adaptive call admission control in circuit switched networks*, IEEE Transactions on Automatic Control, (2000). Submitted.

[35] ——, *An on-line 'surrogate problem' methodology for stochastic discrete resource allocation problems*, J. of Optimization Theory and Applications, 108 (2001), pp. 349–376.

[36] W. B. Gong, Y. C. Ho, and W. Zhai, *Stochastic comparison algorithm for discrete optimization with estimation*, Proc. of 31st IEEE Conf. on Decision and Control, (1992), pp. 795–800.

[37] B. Griggs, G. Parnell, and L. Lehmkuhl, *An air mission planning algorithm using decision analysis and mixed integer programming*, Operations Research, 45 (1997).

[38] Y. Guo, *On Fluid Modeling of Networks and Queues*, PhD thesis, University of Massachusetts Amherst, Amherst MA, September 2000.

[39] Y. Guo, X. Yin, and W. Gong, *ART2 neural network clustering for hierarchical simulation*, in Proceedings of 19th SPIE Conference, Orlando, Apr. 1998.

[40] H. Hafner, *Lot-sizing and throughput times in a job shop*, International Journal of Production Economics, 23 (1991), pp. 111–116.

[41] Y. Ho and X. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems*, Kluwer Academic Publishers, Boston, Massachusetts, 1991.

[42] Y. C. Ho, R. S. Sreenivas, and P. Vakili, *Ordinal optimization in DEDS*, J. of Discrete Event Dynamic Systems: Theory and Applications, 2 (1992), pp. 61–88.

[43] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[44] T. Ibaraki and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, Cambridge, MA, 1988.

[45] G. Jablunovsky, C. Dorman, and P. Yaworsky, *A neural network sub-model as an abstraction tool: Relating network performance to combat outcome*, in Proceedings of SPIE, Orlando, Florida, Apr 2000.

[46] G. Kaplan, *Simulating networks*, IEEE Spectrum, (2001), pp. 74–75.

[47] G. Kesidis, A. Singh, D. Cheung, and W. Kwok, *Feasibility of fluid-driven simulation for ATM network*, in Proceedings of IEEE GLOBECOM, vol. 3, 1996, pp. 2013–2017.

[48] H. Kobayashi and Q. Ren, *A mathematical theory for transient analysis of communications networks*, IEICE Transactions on Communications, E75-B (1992), pp. 1266–1276.

[49] K. Kumaran and D. Mitra, *Performance and fluid simulations of a novel shared buffer management system*, in Proceedings of IEEE INFOCOM, Mar 1998.

[50] H. Kushner and D. Clark, *Stochastic Approximation for Constrained and Unconstrained Systems*, Springer-Verlag, Berlin, Germany, 1978.

[51] T. Lane, *Hidden Markov Models for human/computer interface modeling*, in IJCAI-99 Workshop on Learning About Users, 1999, pp. 35–44.

[52] T. Lane and C. E. Brodley, *Temporal sequence learning and data reduction for anomaly detection*, ACM Transactions on Information and System Security, 2 (1999), pp. 295–331.

[53] C. E. Lau, *Neural Networks: Theoretical Foundations and Analysis*, IEEE Press, 1991.

[54] S. Lee and T. Tang, *Transport coefficients for a silicon hydrodynamic model extracted from inhomogeneous Monte Carlo calculations*, Solid-State Electronics, 35 (1992), pp. 561–569.

[55] B. Liu, D. Figueiredo, Y. Guo, J. Kurose, and D. Towsley, *A study of networks simulation efficiency: fluid simulation vs. packet-level simulation*, in Proceedings of IEEE Infocom 2001, vol. 3, Alaska, April 2001, pp. 1244–1253.

[56] B. Liu, Y. Guo, J. Kurose, D. Towsley, and W. Gong, *Fluid simulation of large scale networks: Issues and tradeoffs*, in Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, Jun 1999.

[57] Y. Liu and W. Gong, *Perturbation analysis for stochastic fluid queueing systems*, in Proc. 38th IEEE Conf. Dec. and Ctrl, 1999, pp. 4440–4445.

[58] J. Marin, D. Ragsdale, and J. Surdu, *A hybrid approach to the profile creation and intrusion detection*, in Proceedings of DARPA Information Survivability Conference and Exposition, Anaheim, CA, 2001.

[59] K. G. Mehrotra, C. K. Mohan, and S. Ranka, *Bounds on the number of samples needed for neural learning*, IEEE Transactions on Neural Networks, 2 (1991), pp. 548–558.

[60] S. Meyn, *Sequencing and routing in multiclass networks. Part I: Feedback regulation*, in Proceedings of the IEEE International Symposium on Information Theory, 2000, pp. 4440–4445. To appear in SIAM J. Control and Optimization.

[61] V. Misra and W. Gong, *A hierarchical model for teletraffic*, in Proc. 37th IEEE Conf. Decision and Control, vol. 2, Tampa FL, 1998, pp. 1674–1679.

[62] N. Miyoshi, *Sensitivity estimation of the cell-delay in the leaky bucket traffic filter with stationary gradual input*, in Proceedings of the International Workshop on Discrete Event Systems, WoDES'98, Cagliari, Italy, Aug 1998, pp. 190–195.

[63] B. Mohanty and C. Cassandras, *The effect of model uncertainty on some optimal routing problems*, Journal of Optimization Theory and Applications, 77 (1993), pp. 257–290.

[64] D. Nicol, M. Goldsby, and M. Johnson, *Fluid-based simulation of communication networks using SSF*, in Proceedings 1999 European Simulation Multiconference, October 1999.

[65] *http://www.cc.gatech.edu/computing/compass/pdns/index.html.*

[66] C. Panayiotou, C. Cassandras, and W.-B. Gong, *Model abstraction for discrete event systems using neural networks and sensitivity information*, in Proceedings of the Winter Simulation Conference, Dec. 2000, pp. 335–341.

[67] C. G. Panayiotou and C. G. Cassandras, *Optimization of kanban-based manufacturing systems*, Automatica, 35 (1999), pp. 1521–1533.

[68] R. Parker and R. Rardin, *Discrete Optimization*, Academic Press, Inc, Boston, 1988.

[69] L. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Proceedings of IEEE, 77 (1989), pp. 267–293.

[70] M. I. Reis dos Santos and A. M. O. Porta Nova, *The main issues in nonlinear simulation metamodel estimation*, in Proceedings of the Winter Simulation Conference, Dec 1999, pp. 502–509.

[71] R. Rubinstein and A. Shapiro, *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*, John Wiley and Sons, New York, New York, 1993.

[72] A. Seleznyov and S. Puuronen, *Anomaly intrusion detection systems: Handling temporal relations between events.* Recent Advances in Intrusion Detection (RAID'99), 1999.

[73] L. Shi and S. Olafsson, *Nested partitions method for global optimization*, Operations Research, 48 (2000), pp. 390–407.

[74] *http://www.ssfnet.org.*

[75] H. Takahashi and H. Gu, *A tight bound on concept learning*, IEEE Transactions on Neural Networks, 9 (1998), pp. 1191–1202.

[76] U.S. Army Concept Analysis Agency, *Concept Evaluation Model*, 1983.

[77] Y. Wardi and B. Melamed, *IPA gradient estimation for the loss volume in continuous flow models*, in Proceedings of the International Workshop on New Directions of Control and Manufacturing, Hong Kong, Nov 1994, pp. 30–33.

[78] ——, *Continuous flow models: Modeling, simulation and continuity properties*, in Proceedings of the 38th IEEE Conference On Decision and Control, Phoenix, Arizona, Dec 7-10 1999, pp. 34–39.

[79] ——, *Loss volume in continuous flow models: Fast simulation and sensitivity analysis via IPA*, in Proceedings of the 8-th IEEE Mediterranean Conference on Control and Automation (MED 2000), Patras, Greece, Jul 17-19 2000.

[80] ——, *Variational bounds and sensitivity analysis of traffic processes in continuous flow models*, Discrete Event Dynamic Systems: Theory and Applications, 11 (2001), pp. 249–282.

[81] C. Warrender, S. Forrest, and B. A. Pearlmutter, *Detecting intrusions using system calls: Alternative data models*, in IEEE Symposium on Security and Privacy, 1999, pp. 133–145.

[82] J. Wieselthier, C. Barnhart, and A. Ephremides, *Standard clock simulation and ordinal optimization applied to admission control in integrated communication networks*, Journal of Discrete Event Dynamic Systems, 5 (1995), pp. 243–279.

[83] E. Wong, *Stochastic neural networks*, Algorithmica, 6 (1991), pp. 466–478.

[84] Y. Wu and W. Gong, *Time stepped simulation of queueing systems*, in Proceedings of SPIE, vol. 4367, April 2001, pp. 262–273.

[85] A. Yan, *On Some Modeling Issues In High Speed Networks*, PhD thesis, University of Massachusetts Amherst, Amherst MA, February 1998.

[86] A. Yan and W. Gong, *Fluid simulation for high-speed networks with flow-based routing*, IEEE Transactions on Information Theory, 45 (1999), pp. 1588–1599.

[87] D. Yan and H. Mukai, *Stochastic discrete optimization*, SIAM Journal on Control and Optimization, 30 (1992), pp. 549–612.

[88] M. A. Zeimer and J. Tew, *Metamodel applications using TERSM*, in Proceedings of the 1995 Winter Simulation Conference, Dec. 1995, pp. 1421–1428.